A DISTRIBUTED SYSTEM FOR MULTISCALE ANALYSIS AND VISUALIZATION OF LARGE-SCALE AIRBORNE LIDAR POINT CLOUDS

Satendra Singh

Master of Science by Research Thesis

November 2021

International Institute of Information Technology, Bangalore

A DISTRIBUTED SYSTEM FOR MULTISCALE ANALYSIS AND VISUALIZATION OF LARGE-SCALE AIRBORNE LIDAR POINT CLOUDS

Submitted to International Institute of Information Technology,
Bangalore
in Partial Fulfillment of
the Requirements for the Award of
Master of Science by Research

by

Satendra Singh
MS2017004

International Institute of Information Technology, Bangalore
November 2021

Dedicated to

This thesis is dedicated to my family and wife, Manjari Singh, for their endless support and encouragement.

Thesis Certificate

This is to certify that the thesis titled **A Distributed System for Multiscale Analysis and Visualization of Large-scale Airborne LiDAR Point Clouds** submitted to the International Institute of Information Technology, Bangalore, for the award of the degree of **Master of Science by Research** is a bona fide record of the research work done by **Satendra Singh**, **MS2017004**, under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

_____

Prof. Jaya Sreevalsan Nair

Bangalore,

The 29th of November, 2021.

A DISTRIBUTED SYSTEM FOR MULTISCALE ANALYSIS AND VISUALIZATION OF LARGE-SCALE AIRBORNE LIDAR POINT CLOUDS

Abstract

Several approaches exist in literature and practice, for processing and visualizing large-scale point clouds. Data management has been done using spatial databases. The semantic classification has been demonstrated successfully using both supervised and deep learning methods in 3D large-scale Light Detection and Ranging (LiDAR) point cloud of urban regions. Urban regions particularly have multi-class data. Local neighborhoods are used for feature extraction in these point clouds. Also, massive point cloud visualization is done using the Potree tool. However, these approaches suffer from being available across several systems. The lack of an integrated system brings along challenges in managing and transferring data across these systems to accomplish analytical tasks. Communication across such systems is not scalable with data size.

The expectation from an integrated system is to process, manage, and visualize large-scale data sets, to support exploratory analysis, and to build contextual understanding. We find that the state-of-the-art is predominantly restricted to solving the computational complexity, and the gap in integrating the back-end efficient storage with the computational system continues to exist. In this work, we propose an integrated system to address the gap. We approach this problem by building an integrated big data system using Apache Spark and Cassandra to process and manage, classify and visualize. Standard browser based tool using WebGL can be deployed in the system to interactively visualize the massive point cloud. Such a system will enable to run of experiments involving large-scale data to perform exploratory tasks and contextual

understanding. To demonstrate the usability of the system, we have implemented the computation of local geometric descriptors, feature vectors, and semantic classification using the supervised learning methods. These methods have been implemented using Spark ML library, powered by Cassandra to save the intermediate results. We further interactively visualize these results on the browser, along with performing analytical tasks on the results persisted in Cassandra.

For implementing feature extraction using the integrated system, we make specific design choices in terms of spatial partitioning of the point cloud data, the local neighborhood search, multi-scale method for feature extraction, feature vectors for classification, the classifier itself. The use of Apache Spark also provides the implementation of classifiers using supervised learning through the Spark ML library. We have performed semantic classification of one of the largest annotated benchmark airborne LiDAR dataset, DALES, using our system. The overall accuracy of our classification results using different multiscale feature extraction methods, namely, averaged features and features at optimal scales, have been 85% and 78%, respectively. We propose an adaptive multiscale feature extraction method to further improve the classification outcomes. Our proposed method entails voxelization of point clouds to exploit spatial locality.

We propose the design of a browser-based interactive visualization tool for large-scale airborne LiDAR point clouds that fits with the integrated system. The visual tool achieves the data analytical operations on the user-selected region of interest using the Spark engine backed by the persistence layer. The analytical tasks include determining object class distribution, geometric class distribution, comparison of classification results with the ground truth, wherever applicable.

In summary, we demonstrate the implementation of our proposed integrated system for large-scale point cloud analysis, including machine learning and visualization, using the Spark-Cassandra integration, that is horizontally scalable with fault tolerance.

Acknowledgements

I would like to express my sincere gratitude to my research supervisor, Professor Jaya Sreevalsan Nair, for enabling me to pursue research and providing invaluable advice, continuous encouragement, and support throughout this research work. It has been a great privilege and honor to study and work under her guidance. I would like to thank my fellow members of the Graphics-Visualization-Computing Lab (GVCL), especially Reddy Rani Vangimalla and Pragyan Mohapatra, for their continued support and knowledge sharing.

I would like to thanks all of the authors whose work I have been able to build on.

I would like to thank my family and friends for their invaluable moral support and encouragement which helped me to stay focused on my study.

I would like to thank my current organization Bluebear Technologies Pvt. Ltd., for enabling me to pursue my research and work.

— Satendra Singh

List of Publications

[I] **S. Singh**, and J. Sreevalsan-Nair, "A Distributed System for Multiscale Feature Extraction and Semantic Classification of Large-scale LiDAR Point Clouds," in the Proceedings of the 2020 IEEE India Geoscience and Remote Sensing Symposium (InGARSS), pp 74-77, December 2020. doi : https://doi.org/10.1109/InGARSS48198.2020.9358938.

[II] **S. Singh**, "A Distributed System for Optimal Scale Feature Extraction and Semantic Classification of Large-scale Airborne LiDAR Point Clouds," in Distributed Computing and Internet Technology, Proceedings of the 17th International Conference on Distributed Computing and Internet Technology (ICDCIT), January 2021, Sequence Number 18, Lecture Notes in Computer Science, Springer International Publishing. doi : https://doi.org/10.1007/978-3-030-65621-8_18.

[III] **S. Singh** and J. Sreevalsan-Nair, "Adaptive Multiscale Feature Extraction in a Distributed System for Semantic Classification of Airborne LiDAR Point Clouds," in IEEE Geoscience and Remote Sensing Letters (Early Access), published online July 2021. doi : https://doi.org/10.1109/LGRS.2021.3099935.

[IV] J. Sreevalsan-Nair, P. Mohapatra, and **S. Singh**, "IMGD: Image-based Multiscale Global Descriptors of Airborne LIDAR Point Clouds Used for Comparative Analysis," in Smart Tools and Apps for Graphics (STAG 2021) - Eurographics Italian Chapter Conference, P. Frosini, D. Giorgi, S. Melzi, and E. Rodolá, Eds., The Eurographics Association, 2021, pp. 61–72, ISBN : 978-3-03868-165-6. doi : https://doi.org/10.2312/stag.20211475.

Contents

List of Figures

List of Tables

List of Abbreviations

API . . . . . Application Programming Interface

ALS . . . . . Airborne Laser Scanning

AWS . . . . . Amazon Web Services

CPU . . . . . Central Processing Unit

DAG . . . . . Directed Acyclic Graph

GBT . . . . . Gradient-Boosted Trees

GPU . . . . . Graphics Processing Unit

GUI . . . . . Graphical User Interface

IO . . . . . Input Output

IOU . . . . . Intersection Over Union

JVM . . . . . Java Virtual Machine

LiDAR . . . . . Light Detection and Ranging

LRM . . . . . Logistic Regression Model

MNO . . . . . Modifiable Nested Octree

OA . . . . . Overall Accuracy

OpenMP . . . . . Open Multi-Processing

RDD . . . . . Resilient Distributed Datasets

RFC . . . . . Random Forest Classifier

ROI . . . . . Region Of Interest

VPC . . . . . Virtual Private Cloud

WebGL . . . . . Web Graphics Library

CHAPTER 1

INTRODUCTION

The point cloud data acquired using LiDAR technology represents the topography of large geographical regions. With the advancement in airborne LiDAR technology, remote sensing and aerial laser scanning contributed to an exponential increase in large-scale data collections. However, accommodating the data on a single machine is intractable and requires storage on clusters of machines. Thus, data exploration, visualization, and contextual understanding require efficient distributed data management and processing for large-scale point clouds, which are acquired using airborne LiDAR.

Point cloud processing mainly involves the computation of local geometric descriptors defined on eigenvalue decomposition at each point using the local neighborhood information using multiple scales. Here, we refer to the size of the local neighborhood as the *scale*, which is used as a parameter in the literature for LiDAR point cloud processing [6]. Thus, the value of the scale parameter represents the extent of search for local neighbors of any point. In the case of spherical local neighborhood, the radius of spherical search serves as the scale, and similarly, in the case of K-nearest neighborhood, the scale is the value k used for the number of neighbors to be determined. The contextual understanding is given by the semantic classification of a point cloud, which is a widely used data processing method. Spatial data structures such as octree, quadtree, and kd-tree are the most widely used hierarchical data structure to build

efficient algorithms like K-nearest neighbor or radial search. However, these data structures do not scale with size. The general approach in the point cloud is to split massive datasets into multiple tiles and then process them on a separate machine. To determine local neighborhood information in a distributed environment requires defining the spatial partitioning of data and building new efficient methods for neighborhood search. The neighborhood search for a point entails finding the points that satisfy a specific *local neighborhood* criterion, *e.g.,* located within a specific distance from the point or K-nearest and perform a range search. Hence, the extraction of the local neighborhood in a distributed system is an important problem to study.

The local neighborhood is used to compute local geometric descriptors used for feature extraction for semantic classification and segmentation. These descriptors and features expand the storage requirements as they are required to be available together for further machine learning and visualization tasks. In the case of massive data, it requires efficient distributed data storage management. For instance, in the case of the DALES dataset [5], the features along with location and class label have a training dataset of 8 GB and test data of 3 GB, while the training feature generated requires 70 GB, and the test feature requires 26 GB. The storage requirement in the case of multiscale analysis further increases with the addition of each scale. Building machine learning models using the massive feature dataset certainly requires distributed processing. The state-of-the-art machine learning libraries such as TensorFlow can process such massive feature datasets, but its focus remains on deep learning outcomes, leaving out the intermediate results that are useful for data exploration.

The state-of-the-art supervised learning and classification of points involve finding the optimal set of neighboring points to build the local geometric descriptors and feature vector. We say it is *optimal*, as the set must provide a feature vector for the point, that is capable of disambiguating the semantic class the point belongs to. Since there is inherent uncertainty in environmental data, we extract these feature vectors from

multiple spatial scales. The choice of scales plays an essential role in building features to generate high-accuracy classifiers. Hence, finding the optimal set of scales for each point is a problem that must be tackled for generating an efficient feature vector to train ML models. Our work also suggests improving the classification result using an adaptive method for aggregating information from multiple scales, which gives the most optimal feature vector.

In addition to semantic classification, visual analytics of these massive point clouds is needed for improving the understanding of the data and support exploration. Visualizing local descriptor and classification results details and analytics on top of the ground truth helps contextual understand and refine the features and models. Tools required for visualization are mostly desktop-based, which require the data and the supported libraries to be available on the local machine. Web browser-based tools, such as Potree [7] visualize massive point cloud on a standard browser, but they require conversion of data to the supported file formats before they are ready for visualization.

To perform a complete analysis of the large-scale point cloud, the state-of-the-art technology requires several systems to be deployed. The interoperability of these systems is essential to best utilize the available resources efficiently and effectively for the specific processing needs. The data format and the source required by one system need to be made available by the other system. It poses two major challenges. Firstly, the overall cost to perform analysis of the large-scale point cloud data set increases with the addition of systems for specific tasks. Secondly, the time and data transformation from one format and source to another become complex to manage; any change at any step needs to repeats the whole activity. Hence, we propose the need for a single distributed system that integrates subsystems for processing large-scale point clouds. Here, we propose an integrated system using Spark for processing data to generate descriptors and features, Cassandra to manage data storage, and SparkML to perform semantic classification.

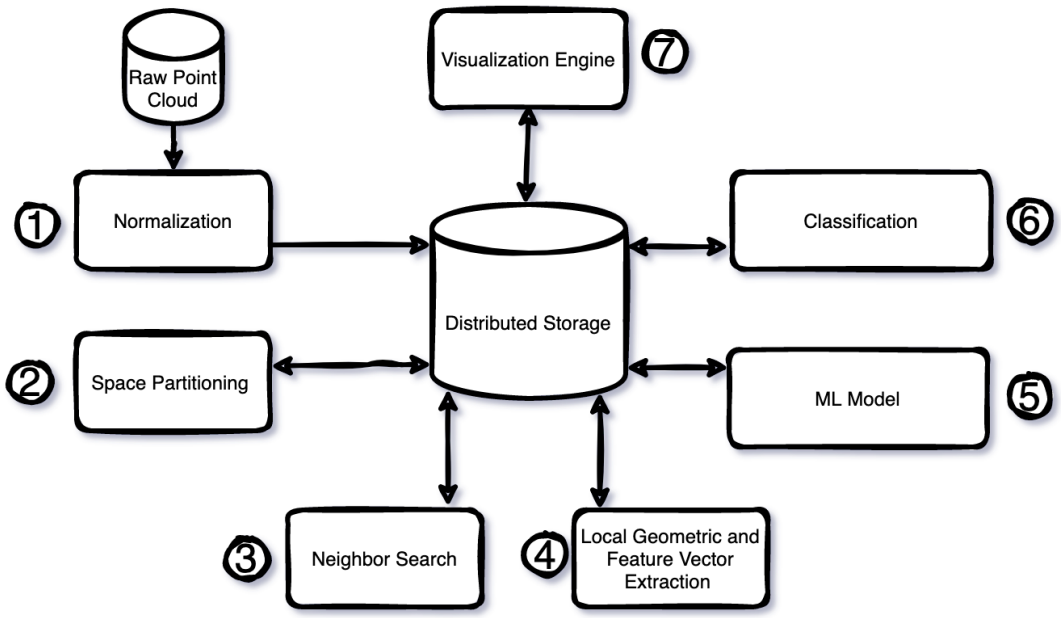

Figure FC1.1: Summary of our proposed solution, *i.e.,* a distributed system built for large-scale LiDAR point cloud processing to support data exploration, visualization and contextual understanding.

1.1 Problem Statement

The search for the neighbor of a point and subsequent extraction of local geometric information precede semantic classification, exploratory analysis, and visualization, in a data science workflow. The entire processing of point cloud, thus, consists of normalization, spatial partitioning, neighbor search, local geometric descriptor and feature vector extraction, semantic classification, and visualization, as shown in Figure FC1.1, where arrows indicate the flow of data between the processes.

We use big data tools to define methods and process large-scale point clouds due to the increasing scale of point cloud data and the requirement of out-of-core processing solutions. Our objective is to address the following problems:

- A single system to process, manage, classify and visualize large-scale point cloud

which is scalable and fault-torelance.

- Various large-scale point cloud analysis task requires a specific system for each task. The availability of each system requires its own management, which makes it a cumbersome activity to perform the sequence of tasks involving multiple systems. Hence, the system is required to employ one integrated system to solve many problems, allowing researchers and professionals to focus on solving their data problems instead of learning and maintaining different systems for each activity.

- Implement partition strategy and neighbor search method for large-scale datasets.

  - Various attempts have been made recently to efficiently define the neighbor search algorithms like K-nearest or range search using big data tools like Hadoop and Spark, which are built on the concept of map and reduce involving data partitioning. However, as the dataset size increases and is partitioned across different nodes, the communication between nodes to find query results in the entire dataset increases latency. Thus, there is a need to build a simple and efficient partitioning strategy to distribute the data evenly as well as to minimize latency in neighbor search.

- Implement multiscale feature extraction and semantic classification using the distributed system.

  - The most widely used method to understand the contextual understanding of point cloud data is semantic classification. The extraction of efficient features eventually allows building the most effective ML models. Given the uncertainty in environmental datasets, multiscale methods have been used for improving the efficiency of extracted features for classification. We design appropriate workflows

in the distributed system to facilitate multiscale feature extraction for semantic classification.

- Improve the semantic classification using the adaptive method.

  - The ensemble model combines the decision from multiple models to improve the overall classifier model. Ensembles primarily use multiple models outcome based on the feature attributes. Similarly, the outcome of the models depends on the features crafted and can give varying performance owing to the object being classified. Optimal scale determination based on Shannon entropy and feature extraction at the optimal scale is a widely used multiscale feature extraction method. Alternatively, averaging features across multiple scales can be used to aggregate information in the different scales. The question on which multiscale feature extraction method is the best for any given point still remains. Hence, we explore the adaptive choice of the multiscale feature extraction method to improve classification accuracy.

- Implement interactive visualization and analytics on the raw and processed point cloud data.

  - The exploratory analysis supported by visualization enables the contextual understanding, which in turn refines the processes shown in Figure FC1.1. Hence, we design an interactive visual analytics tool to improve our feature extraction and semantic classification.

The main contribution of this thesis is to study big data tools fit for large-scale processing and provide the integrated framework to allow running various point cloud analyses and experiments.

1.2 Our Contributions

We propose a distributed system using the Apache Spark and Cassandra for processing large-scale point clouds. Our system design includes appropriate partitioning of the data, extraction of the neighborhood of a point, building the optimal and average scale feature vector, performing semantic classification, and storing the results of each task in the distributed data store. We then visualize and interactively perform the analytics to improve the semantic classification. The analytics demonstrate the comparison between the ground truth class and classification outcomes, in the form of distributions computed from the previously stored result.

- The novelty of our work lies in adopting the Cassandra as the distributed storage along with the Spark engine to make the data locally managed using the Spark-Cassandra connector. It improves the overall input-output (IO) and performance and achieves one single system's goal to perform various tasks.
- We define an alternative way to partition the space evenly to reduce the complexity and define the efficient neighbor search algorithm by approximating the spherical search. We also consider retaining the boundary points not to miss any information.
- We focus on generating two types of multiscale local geometric descriptors for feature extraction, *i.e.,* averaged feature vector across scales and feature vector at optimal scale. These features are subsequently used for semantic classification. The choice of multiscale method is made towards improving the classification results.
- As the feature vectors are available, we experiment with different attributes to analyze the significance of the features on model efficiency.

- We discuss different classification models using supervised classifiers. We also establish the impact of retaining the boundary point while creating the partition on the chosen classifiers.
- To further improve the classification accuracy, we use a control strategy to make an adaptive choice between optimal and averaged features at each point. We then propose an efficient data management strategy to implement multiscale aggregation using point cloud voxelization and logistic regression.
- We demonstrate our visualization tool on a standard web browser to perform real-time analytics. The contribution includes defining the sub-sampling of the data for different resolutions using the Spark engine and Cassandra. We also built analytics features in the tool using the Spark engine and visualize it for comparative analysis with ground truth.
- Our work also discusses the limitation of the proposed system, various improvements and the scope of future work.

1.3 Thesis Structure

Chapter 2 gives the overview of related work, including the previous works to process large-scale point cloud data, and work related to visualize the massive point clouds on a standard web browser. Chapter 3 provides the detailed description of generic system architecture built using big data tool, its components, and how we exploit the custom partitioning, data locality using Spark and Cassandra. Chapter 4 discusses the strategy to partition the point cloud, local descriptor generation, and classification using the Spark-Cassandra cluster. Chapter 5 discusses the adaptive strategy to improve the classification using the voxel level classifier selection. Chapter 6 discusses the visualization system to render massive point cloud and interactions with analytics widgets.

Chapter 7 concludes the thesis with the future work.

CHAPTER 2

RELATED WORK

This chapter will give an overview of the previous work and ongoing research related to large-scale point cloud data processing, management, semantic classification, and interactive visualization.

2.1 Processing Large-scale Point Cloud

Most of the efficient algorithm exists to process point cloud are built on space partition data structures like octree, kd-tree or quadtree. Behley *et al.* [8] have improved radius neighbor search in three-dimensional point clouds using efficient data structures. However, data structure generation does not scale with the size of point clouds. The data structures do not fit in the main memory for large-scale point clouds, thus requiring out of core technology to store, process, and render even a subset of the data. Hence, one of the essential steps in processing large-scale point clouds in a distributed environment is partitioning the data logically to allow the algorithms to work in parallel on the data.

Boehm *et al.* [9] have used Apache Spark to use the sequence of transformation on a Resilient Distributed Dataset (RDD), which is a partition of fault-tolerant collection of objects, for extraction of tree crowns for the large-scale point cloud. Pajić *et al.* [10] have reduced the dimensionality of the point cloud using the space-filling curve, Z-

Curve, to 1D. They then extended Apache Spark DataFrame to determine the k-nearest neighborhood and range query search of massive point cloud data set.

2.2 Data Management of Large-scale Point Cloud

A large-scale point cloud needs efficient storage and querying performance both. The traditional methods were primarily using a file system to manage. The technique involves compression of data and a specific file format to store the data, and algorithms were built to read in applications. Pavlovic *et al.* [11] have explored the use of an in-memory database, namely SAP HANA, for large-scale point cloud management, supported by indexing using space-filling curve dictionary-based compression. Pajić *et al.* [10] have demonstrated the comparison of the query performance between Postgress DB and Spark SQL on large-scale data.

2.3 Semantic Classification

Weinmann *et al.* [12] have performed extensive experiments, showcasing the effectiveness of supervised learning methods in point cloud classification, given the uncertainty in the data. The feature vector is obtained from the raw values and local geometric descriptor [6,13] computed at each point using information from multiple scales. Multi-scale feature extraction is an approach by which the feature vector is computed at each scale and then aggregated. Usually, the final feature vector is either the average of the feature vectors at different scales [13] or is the one at the optimal scale [6]. The latter is an adaptive method. However, as the number of scales increases, the generation of descriptors becomes compute-intensive. In recent work, Shannon entropy is computed from eigenvalues of the covariance matrix representing the local neighborhood at the scale to find the optimal scale to build the feature vector for model learning. The fea-

ture vector at the optimal scale is used for classification [6], and has been found to be effective [12].

Hackel *et al.* [14] have explored parallel processing using Open Multi-Processing (OpenMP) and deep learning framework, Torch, to perform classification of Semantic3D data using the random forest classifier, where Semantic3D [15] is one of the large-scale annotated data sets.

2.4 Interactive Visualization of Large-scale Point Cloud

Managing large-scale point cloud for rendering does not fit in main memory and scale well and requires data partition and out of core process and render the only subset of the data at a time. Markus Schütz [7] has developed Potree, a visualization tool for large-scale point clouds. Potree uses an improved version of the modifiable nested octree (MNO) [16] to incorporate the grid approach instead of the nested octree. This data structure stores subsamples of the original point cloud in each node, whereas the size of the node decreases and the density increases as the level increases. In addition, this approach allowed the zoom-in feature on the graphical user interface (GUI) to visualize points in more detail. The data structure is stored in multiple files, one for each node.

2.4.1 Rendering on the Web

Potree is a popular browser-based rendering tool powered by a backend server that uses MNO data structure to manage and supply data on demand for different resolution requirements and rendering on the web using Web Graphics Library (WebGL). The data used for rendered is reducing using the Poisson-disk subsampling method.

Plasio [17] project is an example of a tool that uses WebGL to render massive point cloud on a standard web browser. It includes an inbuilt Laz decompressor on

the browser to extract the Laz files.

CesiumJS [18], is a JavaScript library for creating 3D globes and 2D maps in a web browser without a plugin. It uses WebGL for hardware-accelerated graphics and is cross-platform, cross-browser, and tuned for dynamic-data visualization. It uses the same octree concept that Potree uses.

2.5 Summary

While Apache Spark has been used for large-scale point cloud processing and Potree renders large-scale point clouds, there is a gap in the system architecture that combines processing, classification, and visualization of large-scale point clouds. Our thesis bridges this gap.

CHAPTER 3

SYSTEM ARCHITECTURE

To improve the performance of processing large-scale point cloud as the data increases, distributed processing solutions become necessary, and that too, with horizontal scalability having high fault tolerance. It is known that large-scale data processing requires a processing framework which can perform tasks quickly by distributing the data and associated processing tasks across multiple computers in a clustered network. Such a distributed framework should also provide the efficient management and monitoring of the load of each cluster. We propose a distributed system that uses Apache Spark and Cassandra as the big data tools to appropriately manage and process the large-scale point clouds. The proposed system consists of mainly three components i.e., Apache Spark, Cassandra and the Spark Cassandra Connector. Apache Spark serves as the engine for large-scale data processing and Cassandra serves as the large-scale data storage and retrieval of the data, partitioned across multiple nodes in a cluster. The Spark Cassandra Connector is optimized for data locality to read and write data in Cassandra, which ensures the data needed by the spark to perform the processing is locally available on the same node in the cluster.

3.1 Apache Spark

A unified analytics engine is designed for large-scale distributed data processing on-premises in data centers or the cloud. Spark provides in-memory storage for intermediate computations, making it much faster than Hadoop MapReduce. It builds its query computations as a directed acyclic graph (DAG); its DAG scheduler and query optimizer construct an efficient computational graph that can usually be decomposed into tasks executed in parallel across workers on the cluster. Its physical execution engine, Tungsten, uses whole-stage code generation to generate compact code for execution. It incorporates libraries with composable Application Programming Interface (API) for machine learning (MLlib), SQL for interactive queries (Spark SQL), stream processing (Structured Streaming) for interacting with real-time data, and graph processing (GraphX).

Spark delivers easy to use framework by using a simple logical data structure called RDD using which all the other higher-level structured data abstractions, such as DataFrames and Datasets, are built. In addition, by providing a set of transformations and actions as operations, Spark offers a simple programming paradigm that you can use to build big data applications in familiar languages.

Spark supports several programming languages to perform various operations workload: Scala, Java, Python, SQL, and R. Spark offers unified libraries with well-documented APIs that include the following modules as core components: Spark SQL, Spark Structured Streaming, Spark MLlib, and GraphX, combining all the workloads running under one engine.

Spark is built to perform fast parallel computation but does not provide extensive support for storage. Unlike Apache Hadoop, which provides both storage and computation, Spark keeps them decoupled by design. This design philosophy implies that

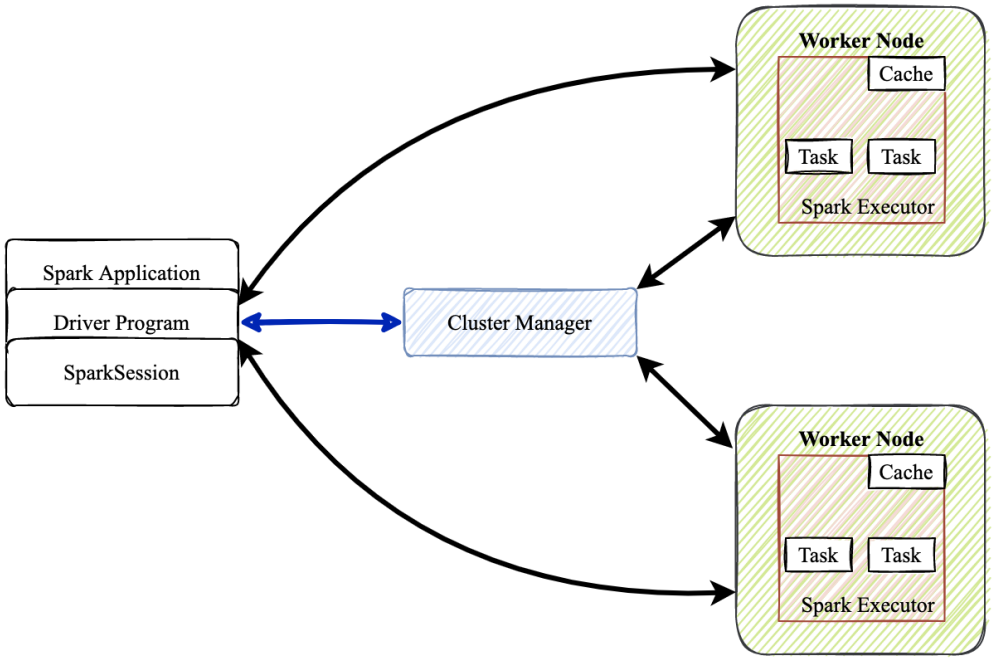

Figure FC3.1: The architecture of the Spark component as used in our proposed system.

Spark can be integrated with storage services to read data stored in multiple sources, *e.g.,* Apache Hadoop, Apache Cassandra, Apache HBase, MongoDB, Apache Hive, RDBMSs, and more. Such integration can then facilitate the processing of all data in memory. The DataFrame Readers and DataFrame Writers in Spark are also extensible to read data from other sources, such as Apache Kafka, Kinesis, Azure Storage, and Amazon S3, into its logical data abstraction, on which it can operate.

Spark is a distributed data processing engine and operates on a cluster of machines. In our implementation, as shown in Figure FC3.1, the Spark application has a driver program as its component that is responsible for managing parallel operations on the Spark cluster. A Spark Session provides the accessibility of the distributed components like cluster manager and executor. Here, we explain each of the terms needed to understand the setup of our Spark component:

- **Spark Driver**: As a part of the Spark application responsible for instantiating

a Spark Session, the Spark Driver plays multiple roles – it communicates with the cluster manager; it requests resources (Central Processing Unit (CPU), memory, etc.) from the cluster manager for the executors, *i.e.,* Java Virtual Machines (JVMs) used in Spark; and it transforms all the Spark operations into DAG computations, schedules them, and distributes their execution as tasks across the Spark executors. Once the resources are allocated, it communicates directly with the executors.

- **Spark Session**: Prior to Spark 2.0, the Spark Context was used as a mechanism to access all the functionalities in Spark. It meant that, in order to connect to each functionality, the application requires the setup of a specific context, *e.g.,* to connect to SQL functionality, SQLContext is needed, and to access the streaming APIs, the streamingContext is needed. However, to build each context, we need to pass the Spark Conf, which contains the cluster information. Now, for Spark 2.0+, Spark sessions encapsulate all these contexts and need not build a separate context to access all the features. Instead, it builds the Spark Context that provides the method to get this context without explicitly creating it. It is the unified entry point for all Spark applications and provides an encapsulated framework to access all spark functionalities with fewer constructs.

- **Cluster Manager**: The Cluster Manager manages and allocates resources required by nodes in the cluster on which the spark application runs. Spark supports four cluster managers: the inbuilt stand-alone cluster manager, Apache Hadoop YARN, Apache Mesos, and Kubernetes.

- **Spark Executor**: A Spark Executor runs on each worker node in the cluster in a Spark application. The executors communicate with the driver program and are responsible for executing tasks on the workers. In most deployment modes, only a single executor runs per node.

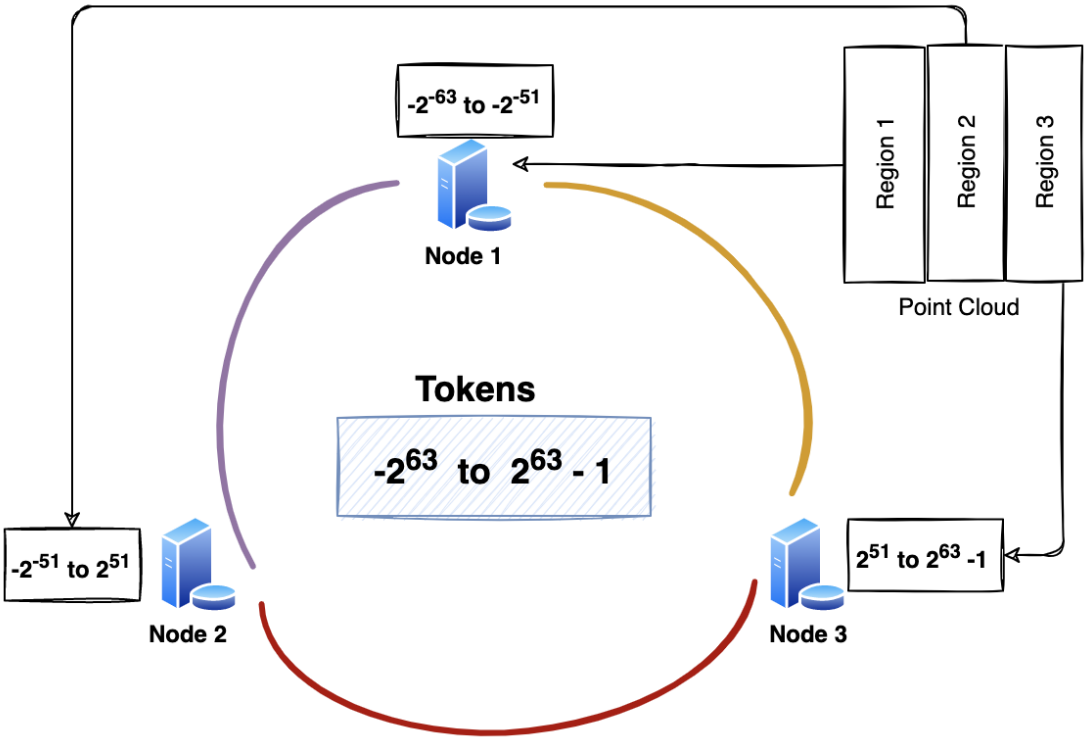

Figure FC3.2: Cassandra nodes with their token ownership, as used in our proposed system.

3.2 Apache Cassandra

Apache Cassandra is an open-source, distributed, NoSQL database. It presents a partitioned wide-column storage model with eventually consistent semantics. It distributes and replicates data across multiple cluster nodes by partitioning all data stored in the system using a hash function. Each partition is replicated to multiple physical nodes, often across failure domains such as racks and even datacentres. Cassandra uses a special form of hashing called consistent hashing to partition data over storage nodes.

Cassandra uses the concept of tokens to distribute data across nodes and perform the indexing. Each node is assigned the range of tokens to hold as shown in Figure FC3.2. The token ID is in the range $[-2^{63}, +2^{63} - 1]$. The Cassandra Partitioner maps the partition key to one of the tokens.

3.2.1 Data Partitioning

Data partitioning is performed using our custom partitioning algorithm, which is configured at the cluster level while the partition key is configured at the table level. Our custom partitioning algorithm is necessary here to preserve spatial locality to reduce the communication latency for neighbor search. Thus, `RegionID` is the ID assigned to the spatial partitions of the point cloud, performed in the context of our application. Here, we discuss how keys are assigned to each partition in Cassandra and how `RegionID` is incorporated with the data partitioning needed in Cassandra. We discuss spatial partitioning in detail in Section 4.1.2.

By design, Cassandra uses *consistent hashing* to distribute and manage the data in the cluster. The *partition keys* are required for managing the consistent hashing partitions. Partition keys are stored in the *primary keys*. The *clustering keys* are needed within the partition.

Primary Key: In our application, we have a table having attributes, such as, [`RegionID`, `X`, `Y`, `Z`, `Label`]. The primary key uniquely defines a row of data in a table. It can be a single attribute or a combination of attributes of the row. Cassandra will perform an upsert operation if we have two or more rows having the same primary key. Examples of the primary key used in our application are `PRIMARY KEY (RegionID, X, Y, Z)` and `PRIMARY KEY (RegionId)`, where the former is a *composite* primary key.

Partition Key: In Cassandra, the data is stored in the partition in the distributed cluster of multiple nodes. The partition key is required to group the data in the same partition. The Cassandra ensures the partition stays in the same node for quick access. The primary and the partition keys are the same for the case of a single attribute. For composite primary, the partition key is the first attribute in the primary key. Just like the primary key can be composite, so can be the partition key. In the *composite* partition key, that

is used for partitioning the data using more than one attribute, the different keys are put within the parenthesis, *e.g.,* `((RegionID, X), Y, Z)`, in this case now the data will be partitioned using the combination of attributes `RegionID` and `X`.

Clustering Key: The data is sorted in the partition using the clustering key before it is stored in Cassandra node. The part of the primary key which is left after removing the partition key is referred to as the clustering key. For example, in primary key `((RegionId, X), Y, Z)` the clustering key will be `Y, Z` and the data will be sorted in the same order of attributes, just as is done in a lexicographical ordering. We can also define the sorting order of each attribute while creating the table in Cassandra by defining the clustering order.

3.2.2 Cassandra Data Modeling and Query Rules

Cassandra is very different from other NoSQL databases, as the data model should be defined based on the query requirements in Cassandra. The objective of data modeling in Cassandra is to write and store data in such a way that can improve the read query. The following rules need to be followed for an effective data model.

1. Spread data evenly around the cluster – the data should be evenly distributed across the cluster. By saying that the choice of partition key should be such that can able to balance between the data and distribute evenly.

2. Minimize the number of partitions read – the objective here is to allow the query to read from one partition and should not be spread across multiple partitions. Even when the partitions lie on the same node, it is complex to read from multiple partitions.

From the above discussion, we find both the rules conflict with each other. As per Rule-2, we should ideally have only one partition for the whole data to reduce the

queries across partitions. But this violates Rule-1. That also means that if we satisfy Rule-1, then Rule-2 will be violated. Overall, we need to balance between the two rules and identify the data model in Cassandra that will optimize the performance.

Cassandra also has several restrictions on the operators that can be used, including:

- The partition key columns support only two operators: = and IN.
- Clustering columns support the =, IN, $>, \geq, \leq, <$, CONTAINS and CONTAINS KEY operators in single-column restrictions and the =, IN, $>, \geq, \leq, <$ operators in multi-column restrictions.

As per the requirements in our application, there are additional constraints to filter the points from one region at a time, and to always execute a query on a single partition.

3.3 Spark Cassandra Connector

The Spark Cassandra Connector performs the table scan in Cassandra and identifies the list of token ranges and knows where these token ranges are hosted in the cluster. Each token range can be used to build the spark partition and we know from which Cassandra node the spark partition build from. Spark driver can then get to know which executor nodes it can assign the partition (CassandraRDD) where Cassandra Node holds the data locally on the node.

The Java Driver running in the Spark Executor pulls rows from the local Cassandra Instance to build the RDD. This imples that the Spark Cassandra Connector ensures that the reading of the data occurs locally, thus, avoiding any network latency and improving the overall performance.

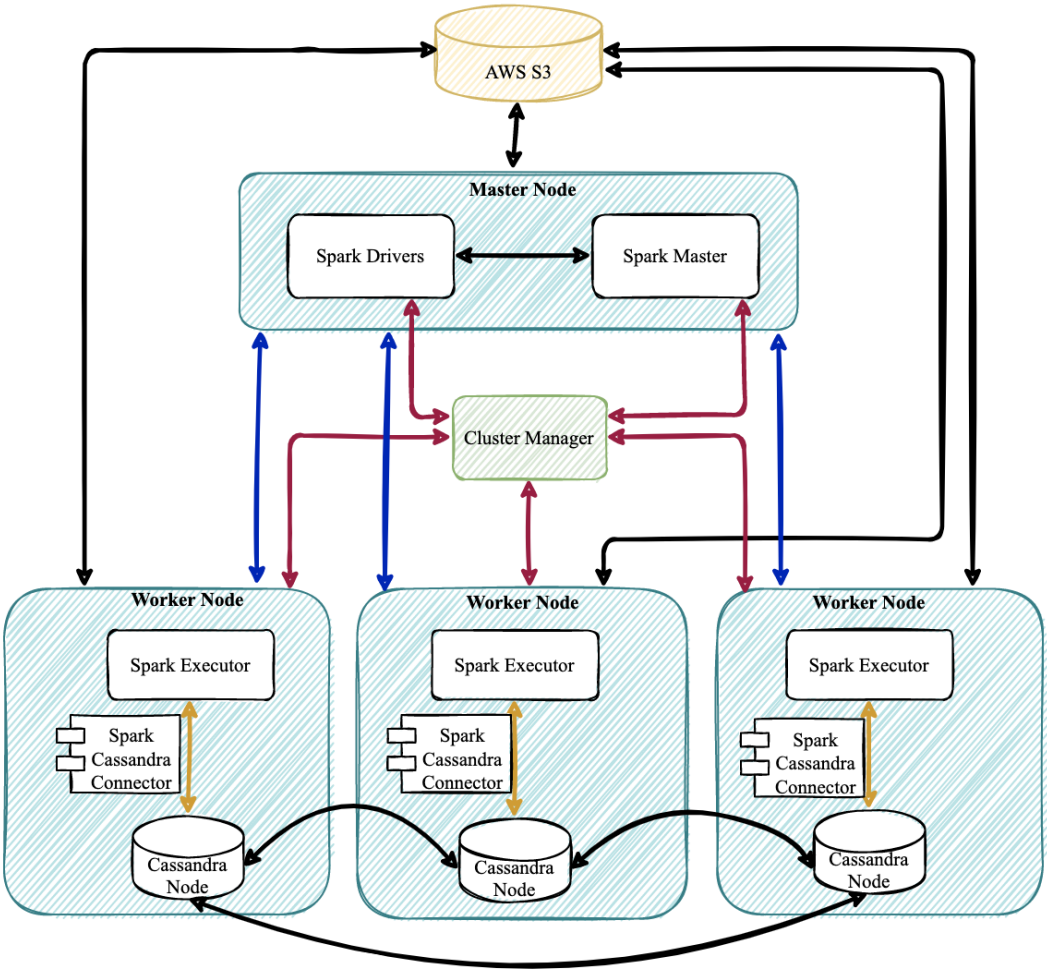

Figure FC3.3: The complete proposed system for large-scale lidar point cloud processing using Apache Spark-Cassandra integration.

3.4 Proposed System Architecture

We deploy the cluster of nodes each running both the Spark worker and Cassandra data node. As Cassandra partitioned the data and persist it across nodes in the cluster, where the partition of the data is determined by the partition key defined on the data. Our Cassandra primary key is defined as `(RegionID, X, Y, Z)` where `RegionID` becomes the partition key and `X,Y,Z` becomes the clustering key.

Figure FC3.3 shows the overall architecture. It stores the raw point cloud in a cloud

computing setup, for which we use the Amazon Web Services (AWS) S3. The total volume of data and number of objects you can store are unlimited. Individual Amazon S3 objects can range in size from a minimum of 0 bytes to a maximum of 5TB. Spark loads the raw point cloud data points from AWS S3 into Spark RDD, and the executor task assigns the `RegionID` to each point. The resultant RDD is stored in Cassandra across cluster nodes using `RegionID` as the partition key. This process implements the spatial partitioning algorithm (Section 4.1.2, partitions our point cloud data into multiple regions, and distributes them across Cassandra cluster nodes and each node stores one region data. Spark applies a custom partitioner using the `RegionID` to partition the data and load it into RDD for processing. This ensures each region's data stays entirely in the local node for processing.

The Spark Executor loads the data from the locally available Cassandra nodes into RDD to process the local data. Any intermediate result, that requires to be persisted, is also stored on locally available Cassandra data node. The determination of the local data source to load in the memory of the Spark Executor is done using the Spark Cassandra Connector library, as shown in Figure FC3.3. Thus, it avoids any IO latency, and also improves the overall performance. Our proposed architecture is designed to be horizontally scalable, thus allowing us to process any data size by just add extra nodes.

3.4.1 Deployment on AWS Cloud

In our implementation, we have used Apache Spark 2.4 and Apache Spark ML, integrated with Cassandra 3.0., with one master node and five executor nodes on Apache Spark. Each of the six nodes uses Intel i7 processor @2.80 GHz, 4 cores, 8 logical processors, 8GB RAM and runs Ubuntu 18.04.1 LTS (GNU Linux 5.4.0-1049-aws) x86_64 operating system. We have additionally used AWS (Amazon Web Services) S3 bucket for storing the classifier model and raw point cloud.

We have created an EC2 image installed with Apache Spark, Cassandra, and java support. We have created the instances from the image and spawned them all in the same private network or a virtual private network, *i.e.,* a virtual private cloud (VPC) in AWS, to reduce the network latency. The S3 bucket has also been created in the same region. To run the applications in the Spark cluster, we have chosen to deploy them using the stand-alone mode. The stand-alone cluster manager mode has also been adopted on top of it. Owing to the horizontally scalable design, whenever the size of the data increases, we need to spawn new instances from the EC2 image, and then add it to the cluster configuration. Thus, our overall architecture is scalable and can be deployed to process unlimited data. In this thesis, we have demonstrated a proof-of-concept of the usability of our system. The performance testing to check how our system performs under certain request load and volume of data is in the scope of future work.

3.5 Summary

In this chapter, we have explained how we have designed and deployed a distributed system using Apache Spark and Cassandra for large-scale point cloud processing. In the following chapters, we discuss its specific functions.

CHAPTER 4

FEATURE EXTRACTION AND SEMANTIC CLASSIFICATION

The 3D topography of large regions can be captured by Airborne LiDAR or aerial laser scanning technology in the form of unstructured large-scale point clouds. In order to allow exploration and generate contextual understanding of large-scale point cloud data requires efficient data management and processing at scale to make sense of the environment and its constituents. Pointwise feature extraction using local geometric descriptors across multiple scales is one of the main processing tasks to achieve object classification. We compute the features at multiple (spatial) scales to address the high uncertainty inherently present in the environmental datasets. The scale refers to the size of the local neighborhood that is used for computing the local geometric descriptor. The local neighbor search that satisfies the condition between the point and the selected scale gives the set of neighbors required for local geometric descriptor computation. The local neighbor search volume is defined based on its shape or the condition to be satisfied. It is known that the search volume type, such as k-nearest neighborhood, cubical, spherical, cylindrical, etc., influences the semantic classification outcome [12].

The multiscale method involves finding the pointwise features at multiple scales, and this information is aggregated appropriately using two different state-of-the-art methods, as explained in this Chapter. We then use these handcrafted features for semantic classification. Supervised learning using ensemble techniques, such as random

forest classifier (RFC), and deep learning techniques have been widely used for semantic classification of LiDAR point clouds. Weinmann *et al.*have given a review of these methods, along with extensive experiments and discussion of results [12], which has been used in the choice of the methods used in this thesis.

Generally, the processes of generating multi-scale descriptors, features, and semantic learning of the point cloud are highly compute-intensive and become a challenging problem for large-scale point cloud data. The key algorithms used to find the local geometric descriptors include searching the local neighborhood points and is a very compute-intensive process. Efficient data structures are being developed and proposed to improve the neighbor search process [8] [19], but these do not scale for large-scale point clouds. The integration of interactive visualization of descriptors, features, and classification of the point cloud also requires efficient data management. The persistence and retrieval of data from the storage is high IO activity and becomes a challenging problem to manage the intermediate results of the large-scale point cloud data. Pavlovic *et al.* [20] have explored using an in-memory database, namely SAP HANA, for large-scale point cloud management, supported by indexing using space-filling curve using Morton code dictionary-based compression.

The existing solutions for this compute-intensive combination do not directly scale for large-scale point clouds in a specific set of data analytics applications, namely, interactive feature visualization and semantic classification.

To achieve the goal, we use our proposed Apache Spark and Cassandra as the big data framework to extract multiscale features and finally generate optimal and average feature pointwise, semantic classification using random forest and gradient boost classifier, and manage the results data at each intermediate state of the processing in Cassandra. We proposed the customized partitioning in Spark and find cubical local neighborhoods for feature extraction. The underline framework is horizontally scal-

able, and theoretically, there is no limit on the size of the data that can be processed.

To implement semantic classification on large-scale point clouds, parallel processing has been adopted. OpenMP [21] and deep learning frameworks, such as Torch [14], have been deployed to perform the classification of Semantic3D using random forest classifiers. k-nearest neighborhood or spherical neighborhood has been a widely used type of local neighborhood with deep learning methods to optimize the performance of these architectures. For instance, Adam optimizer has been used in RandLA-Net [22], which also performs down-sampling of the point cloud on the Graphics Processing Unit (GPU). While parallelized and optimized machine learning frameworks can improve computational efficiency, the big data frameworks have been largely used for both dataset management and processing. Apache Spark has been used for extraction of tree crowns from LiDAR point cloud, using spherical neighborhood [23].

4.1 Feature Extraction Workflow on Distributed System

We propose a three-stage workflow to be implemented on the integrated Apache Spark and Cassandra framework for feature extraction.

- **Stage-1**: Data Preparation – this involves normalization of the data, that is needed for running some of the algorithms in Spark MLlib, and loading the data into Spark and Cassandra Cluster.
- **Stage-2**: Spatial Partitioning Algorithm – this involves the partitioning of the large-scale point cloud data, preserving spatial locality, and assignment of the partition id, and saving the partitions in the Cassandra Cluster, as discussed in Section 3.2.1.
- **Stage-3**: Feature Vector Computation – this involves the selection of features to be implemented on our distributed system and feature extraction using multiple

scales. We choose features from the larger set known to be relevant to the semantic classification of LiDAR point clouds [24, 25].

4.1.1 Stage-1: Data Preparation

In our proposed system (Figure FC3.3), the point cloud is loaded from AWS S3 into the Spark cluster as RDD. We normalize the raw point cloud loaded to be contained inside a cube of size 2 units centered at (0,0,0), without altering its aspect ratio. We then proceed to partition the data into multiple regions along the principal axis.

4.1.2 Stage-2: Spatial Partition Algorithm

As explained in Sections 3.2.1–3.2.2, there is a specific requirement of how the data must be partitioned across the nodes in Cassandra. This also involves the spatial partitioning of the point cloud to ensure that the local neighborhood search queries do not entail inter-node communication. Altogether, strategizing the spatial partitioning of the point cloud is an integral part of our proposed system. As a first cut, we strategize partitioning only along a single axis, in order to reduce the number of partition surface boundaries being created. We also now have to decide the specific shape of the local neighbor search volume. This is because an appropriate choice of the search volume and the spatial partitioning algorithm can improve the efficiency of our system.

We take into consideration the axis having the maximum range in our application, so that the partition sizes will be considerably bigger in comparison to the remaining axes. We choose between x and y axes, and refer to this chosen axis as the *principal axis p*. As shown in Algorithm 1, we use N as the maximum possible spatial contiguous partition (number of regions) along principal axis p, with partition boundaries p_i, for $i = 0, 1, 2, 3, 4, \ldots, N$; where K_i represents a region with index i, only for $i > 0$; l_{max} is the maximum cubical neighbor search scale; $\Delta P = \left(P_{max} - P_{min}\right)$ is the range of data

Algorithm 1 Spatial Partition Algorithm

procedure MAXPARTITIONREGIONS($n, x_{min}, x_{max}, y_{min}, y_{max}, l_{max}$)
    $x_{range} \leftarrow x_{max} - x_{min}$
    $y_{range} \leftarrow y_{max} - y_{min}$

    if $x_{range} \geq y_{range}$ **then**
        $P_{max} \leftarrow x_{max}$
        $P_{min} \leftarrow x_{min}$
    else
        $P_{max} \leftarrow y_{max}$
        $P_{min} \leftarrow y_{min}$

    $\Delta P \leftarrow P_{max} - P_{min}$
    $N \leftarrow \frac{\Delta P}{l_{max} * n}$ ▷ The maximum possible partitions

    $i \leftarrow 0$
    $p[] \leftarrow empty$

    while $i \leq N$ **do**
        $p[i] \leftarrow P_{min} + i * n * l_{max}$
        $i \leftarrow i + 1$
    return N, p

along the principal axis p; and n is the number of Spark worker nodes in the cluster. Thus, we get:

$$N = \frac{\Delta p}{l_{max}.n};\ p_i = p_{min} + i * n * l_{max};\ \text{and for a point } x_p \in K_i,\ K_{i-1} \leq x_p < K_i$$

As shown in Figure FC4.1, for each region K_i, we additionally add the left and right buffer region of size l_{max}. Thus, each partition implies an interval $[p_i - l_{max}, p_i + l_{max}]$. Note that the buffer region only serves the purpose of the availability of the local neighborhood of the boundary points, to avoid the communication latency.

Now that the data is loaded into an RDD in Stage-1, we now assign a `RegionID` to each point. The assignment of `RegionID` is as per our spatial partitioning algorithms to persist the resultant RDD in the Cassandra, as discussed in Section 3.2.1.

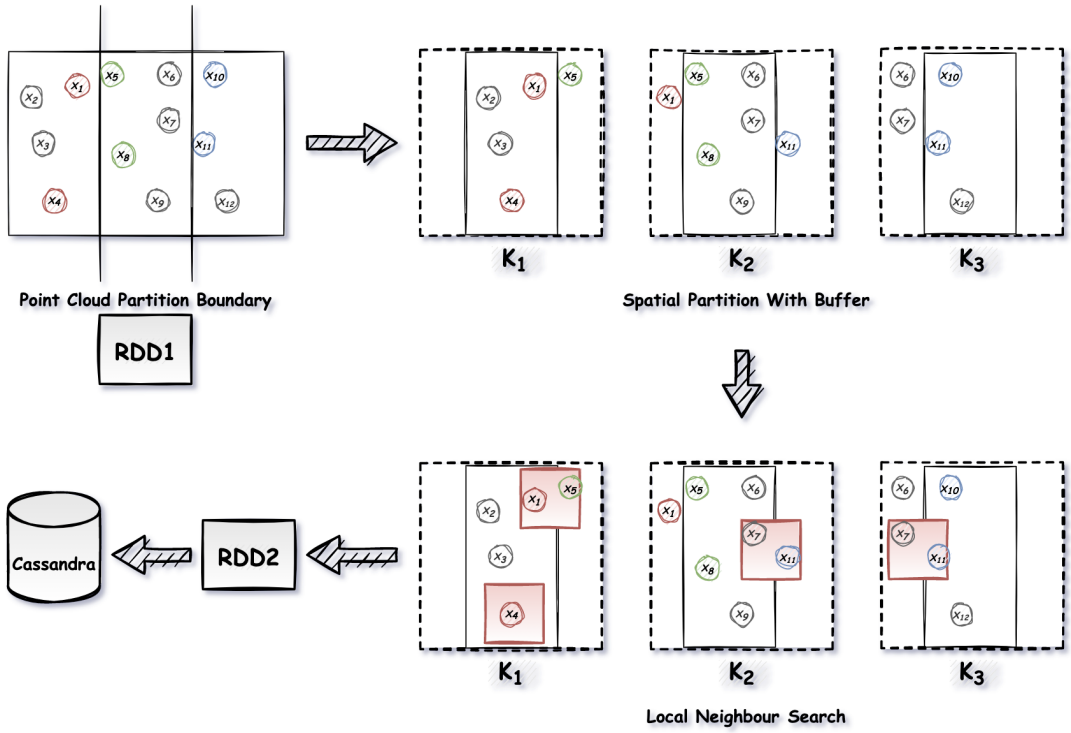

Figure FC4.1: The spatial partition of the point cloud is done strategically such that the each partition is completely resident on a cluster node and it accommodates the neighborhood search queries within the partition itself.

4.1.3 Stage-3: Feature Computation

We load the data from the Cassandra into the Spark cluster by applying the custom partitioner instead of default hash partitioning to ensure that each region exists entirely in a Spark executor node in the form of an RDD, without being split across nodes. Each node can run on the executor per core inside the node. This also implies that a node can load the data of one region per executor. In Figure FC4.2, we see the code example of the custom partitioner where the partition key is the `RegionID` assigned in the previous step of the partition algorithm, and the variable `numParts` is the total count of regions resulting from the spatial partitioning of the point cloud along the principal axis.

The task defined on an RDD is to extract the pointwise features. We implement the task in four steps, in sequence, for each point:

```
public class CustomPartitioner extends Partitioner
{
    private int numParts;

    public CustomPartitioner(int i)
    {
        numParts=i;
    }

    @Override
    public int numPartitions()
    {
        return numParts;
    }

    @Override
    public int getPartition(Object key)
    {
        return ((Long)key).intValue() % numParts;
    }
}
```

Figure FC4.2: An example of the code used for custom partitioner.

- **Step-1**: Local Neighborhood Determination.
- **Step-2**: Local Descriptor Computation.
- **Step-3**: Eigenvalue Decomposition.
- **Step-4**: Feature Computation.

Step-1: Local Neighborhood Determination – Instead of the conventional choice between spherical/radial and k-nearest, we propose to use cubical neighborhood for the local neighbor search volume, as shown in Figure FC4.3. The cubical neighborhood approximates the spherical one, analogous to the use of Chebyshev distance (infinity (L_∞) or maximum norm), instead of Euclidean distance (L_2). This choice of search volume

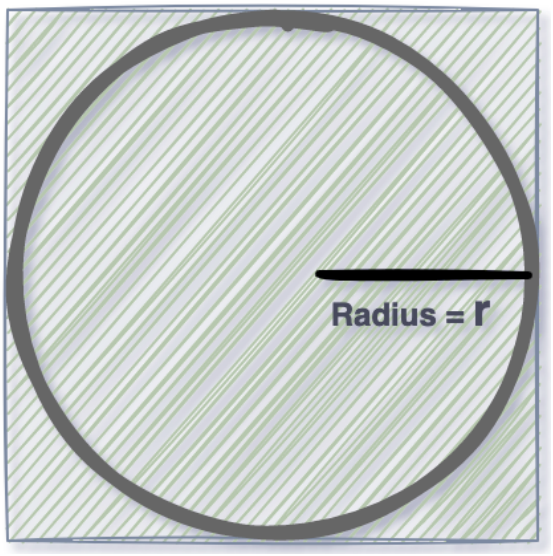

Figure FC4.3: A 2-D analogy of approximating a spherical neighborhood search using its cubical alternative.

type is a key aspect of our strategy in spatial partitioning algorithm (Figure FC4.1) to persist the result and to avoid inter-node communication latencies.

l-cubical neighborhood N_l of a point x in $\mathscr{P}$, such that $\mathscr{P} = \{p \in \mathbb{R}^3\}$, is a set of points which satisfy the Chebyshev distance criterion, $N_l(x) = \{y \in \mathscr{P} \mid \max_{\{0 \leq i < 3\}} (|x_i - y_i|)\}$. Since a spherical neighbourhood of radius r is contained in the cubical neighbourhood of l = 2r, the choice of l is based on the equivalent r that is appropriate for the data set.

Step-2: Local Descriptor Computation – We compute the local geometric descriptor using the neighbors identified in the cubical neighborhood using the covariance tensor T_{3DCM} [26], which is a second-order positive semidefinite tensor [27].

Step-3: Eigenvalue Decomposition – The eigenvalue decomposition of the covariance matrix, *i.e.,* the local geometric descriptor, gives us the eigenvalues such that $\lambda_1 \geq \lambda_2 \geq \lambda_3$. We then compute the saliency map $\{C_l, C_s, C_p\}$, using S=$(\lambda_1 + \lambda_2 + \lambda_3)$, as:

$$C_l = \frac{(\lambda_1 - \lambda_2)}{S}; \qquad C_s = \frac{2(\lambda_1 - \lambda_2)}{S}; \qquad C_p = \frac{3(\lambda_2)}{S}.$$

The saliency map values determine the geometric class of the point [13,27] belongs to surface, line, or point (junction).

Step-4: Feature Vector Computation – We generate the 3D features using the local geometric and the shape properties [28]. The latter includes:

$$\text{Omnivariance, } O_\lambda = \sqrt[3]{\lambda_1 \lambda_2 \lambda_3}$$

$$\text{Eigen-sum, } \sum_\lambda = (\lambda_1 + \lambda_2 + \lambda_3)$$

$$\text{Eigen-entropy or Shannon entropy, } E_\lambda = -\sum_{i=1}^{3} \lambda_i \ln(\lambda_i)$$

$$\text{Change of curvature, } C_\lambda = \lambda_3 / \sum_\lambda$$

$$\text{Linearity, } L_\lambda = (\lambda_1 - \lambda_2)/\lambda_1$$

$$\text{Planarity, } P_\lambda = (\lambda_2 - \lambda_3)/\lambda_1$$

$$\text{Scattering, } S_\lambda = \lambda_3/\lambda_1$$

$$\text{Anisotropy, } A_\lambda = (\lambda_1 - \lambda_3)/\lambda_1$$

These shape features are computed from the eigenvalues of the local geometric descriptors in 3D. Since C_p and C_λ are equivalent, we ignore C_λ. Similarly, the semantics of the saliency map $[C_l, C_s, C_p]$ and the descriptor shape $[L_\lambda, P_\lambda, S_\lambda]$ are the same; thus, we retain the saliency map, $[C_l, C_s, C_p]$, given its relevance in semantic classification in our previous work [1, 2, 29]. We collectively refer to this set of selected eigenvalues as $S_{\lambda,3D}$ [12].

We also generate geometric 3D features. Following are the four features we compute, where the first three are height-based features from local neighborhood, referred to as a set H_{3D}:

- The absolute height z of each point,
- The range Δ_z, *i.e.,* the difference between maxima and minima of height in the local neighborhood,

- The standard deviation σ_z of the height distribution in the local neighborhood of the point,
- The local point density $D = (n_p + 1)/(\frac{4}{3}\pi r^3)$, where n_p is the number of points in the l-cubical neighborhood, and $r = 0.5l$.

We consider the union of feature subset of height-based H_{3D}, and 3D eigenvalue-based features $S_{\lambda,3D}$ [12], and the singleton set of local point density D, as the feature vector. The choice of features is primarily made based on their relevance for classification [25], relatively high classification accuracy with the use of $S_{\lambda,3D}$ at the optimal scale in a random forest classifier (RFC) [12], and the integrability of the feature computation in our distributed system [1, 2]. The feature vector for each scale s, is,

$$\mathrm{F}_s = H_{3D} \cup S_{\lambda,3D} \cup \{D\} = \{E_\lambda, O_\lambda, \sum\lambda, A_\lambda, C_l, C_s, C_p, z, \sigma_z, \Delta_z, D\}.$$

4.1.4 Multiscale Feature Extraction

Computing features at the *optimal* scale is an adaptive multiscale method. The scale where entropy is the global minimum is commonly used as the optimal scale owing to the reduction in uncertainty [6]. While averaging features across scales is an alternative strategy [29], they are contingent upon the averaged features being the optimal representative value across scales. Most of the height- and eigenvalue-based features can generally be averaged across scales to give representative values. The height-based features are computed from the local neighborhood. The local geometric descriptors are positive semidefinite second-order tensors [27], whose tensor invariants are the eigenvalue-based features. Thus, averages can be representative values if there is less statistical variance across scales. Averaging certain features also implies stochastic modeling. The saliency map $\{C_l, C_s, C_p\}$, when averaged, represents the likelihood values of the point belonging to the geometric classes of line, surface, point-type fea-

tures, respectively. The different spatial scales may be considered as mutually exclusive events [27], and the joint probability given by the saliency map is the overall likelihood across scales.

Optimal scale s_{opt} is the scale at which E_λ is the minimum, in which case the point-wise feature vector is $\mathrm{F}_{opt} = \mathrm{F}_{(s=s_{opt})}$. The averaged feature vector across n_s scales is $\mathrm{F}_\mu = \frac{1}{n_s}\sum_s \mathrm{F}_s$.

We also define the size of the neighbourhood l, such that $l_{min} \leq l \leq l_{max}$. Then each scale step size is obtained as : $\Delta l = \frac{(l_{max} - l_{min})}{(N_s - 1)}$.

Thus, the multiscale feature vectors are computed using two different methods, namely, averaging features, and features at the optimal scale. Both sets of feature vectors are then persisted into the Cassandra Spark Cluster using `regionID` as partition key. Also, each vector will have class label included for training and testing for generating and testing the learning models.

4.2 Supervised Classification Models

The feature vectors persisted in the previous steps are used to train the supervised classifier models. Spark MLlib is set of library in Spark ecosystem, which is used to perform machine learning in Apache Spark. We use Random Forest classifier (RFC) and Gradient Boosted Tree classifier (GBT) available in Spark MLlib.

The stored feature vectors in Cassandra are loaded as RDD into the Spark executor. Subsequently, the classifier models are built and stored in S3 file. To test the model, the classifier model is loaded into the spark from the S3 file and then the classifier is run on the test data and final results are stored in the Cassandra for future reporting.

The classification accuracy for the point cloud is measured using intersection over

union (IOU), averaged across all classes, and overall accuracy (OA). For a point cloud with N_c classes,

$$\text{IOU}=\frac{1}{N_c}\sum_{C}\text{IOU}_C, \text{ where for a class, IOU}_C = \frac{TP}{TP+FP+FN},$$
$$\text{and OA}=\sum_{C}\frac{TP+TN}{TP+TN+FP+FN},$$

where TP, TN, FP, and FN are class-wise counts of true-positive, true-negative, false-positive, and false-negative classification outcomes, respectively, for the point cloud.

4.3 Experiments and Results

We have used the distributed system with the specifications, as described in Section 3.4.1. The multiple scales used are $l = 1m, 2m, 3m, 4m, 5m, 6m, 7m, 8m, 9m, 10m$ to generate the features using both methods, *i.e.,* averaged features, and optimal scale features.

Dataset: We have used the recently published Dayton Annotated Laser Earth Scan (DALES) to show the efficacy of the proposed system as shown in Figure FC4.4. DALES is a high-resolution airborne laser scan of the City of Surrey, Canada. The dataset with 0.5 billion points, covering 10 km^2 of area. It contains 40 tiles of dense, labeled point data, including urban regions, of which 29 are training and remaining, testing files. Each tile is of size 0.5 km^2, with a point density of 50 ppm (points per square metre) as average, and 20 ppm as minimum. The eight semantic classes available in DALES are: ground, vegetation, cars, trucks, poles, power lines, fences and buildings. Points that are to be discarded in any learning activity are being labeled as 0.

4.3.1 Optimal Scale Features for Semantic Classification

We have trained the Random Forest Classifier (RFC) on Spark MLlib using ∼51 million points in tiles `5110_54320`, `5110_54460`, `5110_54475`, and `5110_54495` of the

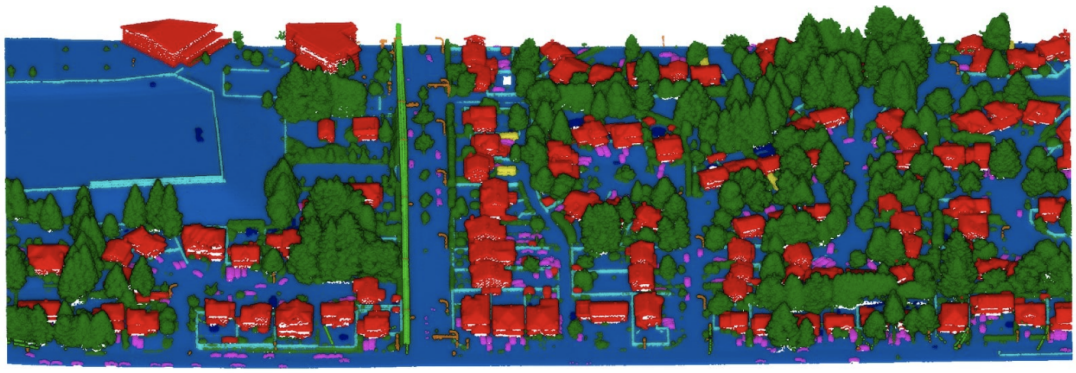

Figure FC4.4: A section of a DALES tile, where the semantic classes are identified by colour; ground (blue), vegetation (dark green), power lines (light green), poles (orange), buildings (red), fences (light blue), trucks (yellow), cars (pink), and unknown (dark blue).

DALES dataset, and tested on other tiles `5080_54470`, `5150_54325` and `5080_54400` each having ∼12 million points from the DALES dataset.

In this experiment, we have also performed the study of the influence of the buffer region. In our case, the buffer region adds up to 10m, on either side of each partition. Thus, it adds the overhead of up to 8K points (∼1.3MB), considering 50 ppm in the data source. This is bound to become significant as the size of the point cloud becomes massive. Nonetheless, our analysis shows that since the local geometric descriptors are additive tensors and by truncating the boundary will provide the local descriptor with a coarser approximation. The results also show that the overall accuracy also did not get drastically impacted, as shown in Table TC4.1.

We have determined the Intersection Over Union (IOU) values for mean, overall accuracy (OA), and per class (Table TC4.1). We have observed an OA of **81.7%** and **79.2%**, with **78.1%** and **74.6%** for the ground class using RFC and GBT respectively. We observe that the absence of buffer region gives us an OA of **79.8%** and **77.3%** with **76%** and **71.9%** for ground class in RFC and GBT case respectively.

For each square tile of 0.5km, the total buffer region with 5 partitions is 0.1km. Thus, we observe that we can have a trade-off of **16%** of additional storage by **2%**

Table TC4.1: Semantic classification result for our case study of DALES point cloud (∼51 million points training, ∼12 million points testing from tile 5080_54470, 11-dimensional feature set) using random forest and gradient boosted tree classifier in a distributed system, using optimal scale features.

(a) Random Forest Classifier with Buffer Region

OA	mean	ground	vegetation	cars
0.817	**0.357**	0.781	0.739	0.154
trucks	powerline	fence	pole	building
0.199	0.238	0.159	0.190	0.395

(b) Random Forest Classifier without Buffer Region

OA	mean	ground	vegetation	cars
0.798	**0.327**	0.760	0.703	0.155
trucks	powerline	fence	pole	building
0.186	0.153	0.134	0.182	0.346

(c) Gradient Boosted Tree Classifier with Buffer Region

OA	mean	ground	vegetation	cars
0.792	**0.351**	0.746	0.626	0.030
trucks	powerline	fence	pole	building
0.030	0.447	0.177	0.206	0.464

(d) Gradient Boosted Tree Classifier without Buffer Region

OA	mean	ground	vegetation	cars
0.773	**0.341**	0.719	0.657	0.041
trucks	powerline	fence	pole	building
0.133	0.487	0.217	0.149	0.321

Table TC4.2: Semantic classification result for our case study of DALES point cloud (∼51 million points training, ∼12 million points testing from tile 5080_54470, 11-dimensional feature set) using different classifiers in our proposed distributed system, using averaged features across multiple scales, using buffer region.

(a) Random Forest Classifier

OA	mean	ground	vegetation	cars
0.836	**0.381**	0.796	0.766	0.208
trucks	power line	fence	pole	building
0.130	0.304	0.227	0.189	0.430

(b) Gradient Boosted Tree Classifier

OA	mean	ground	vegetation	cars
0.817	**0.337**	0.773	0.714	0.063
trucks	power line	fence	pole	building
0.118	0.589	0.117	0.127	0.196

Table TC4.3: Semantic classification result for our case study of DALES point cloud (∼51 million points training, ∼12 million points testing from tile `5110_54325`, 11-dimensional feature set) using Random Forest Classifier in our proposed distributed system, using averaged and optimal features across multiple scales without using buffer region.

(a) Random Forest Classifier Using Average Features

OA	mean	ground	vegetation	cars
0.748	**0.267**	0.397	0.423	0.141
trucks	power line	fence	pole	building
0.171	0.312	0.189	0.226	0.277

(b) Random Forest Classifier Using Optimal Features

OA	mean	ground	vegetation	cars
0.737	**0.242**	0.404	0.394	0.132
trucks	power line	fence	pole	building
0.129	0.390	0.138	0.130	0.218

Table TC4.4: Semantic classification result for our case study of DALES point cloud (∼51 million points training, ∼12 million points testing from tile `5080_54400`, 11-dimensional feature set) using Random Forest Classifier in our proposed distributed system, using averaged and optimal features across multiple scales without using buffer region.

(a) Random Forest Classifier Using Average Features

OA	mean	ground	vegetation	cars
0.8548	**0.3312**	0.831	0.579	0.134
trucks	power line	fence	pole	building
0.267	0.143	0.192	0.107	0.396

(b) Random Forest Classifier Using Optimal Features

OA	mean	ground	vegetation	cars
0.7418	**0.2782**	0.6708	0.481	0.101
trucks	power line	fence	pole	building
0.184	0.141	0.151	0.182	0.315

reduction in overall accuracy in semantic classification.

4.3.2 Averaged Scale Features for Semantic Classification

We have determined the Intersection Over Union (IOU) values for each class, mean IOU, and overall accuracy (OA) in Tables (Table TC4.2), (Table TC4.3) and (Table TC4.4). We have observed an OA of **83.62%**, with IOU of **0.796** and **0.766** for ground and vegetation, respectively, when using RFC as shown in (Table TC4.2). Similar observation can also be seen in (Table TC4.3) and (Table TC4.4) with OA of **74.8%** and **85.48%** tested against tiles `5150_54325` and `5080_54400` respectively. These results are encouraging for a first cut. Accuracy can be further improved by using more scales or more features.

In this experiment, we have additionally run the GBT classifier. Comparatively, the OA for GBT has been lower, as shown in Table TC4.1.

On comparing the RFC results, for the spatial partitioning, including the buffer region, between optimal scale features (Table TC4.1) and averaged features (Table TC4.2), we observe that the latter outperforms the former with a slight margin. Given the minor tradeoff in performance, we analyze the adaptive multiscale approach in Chapter 5 without the buffer region.

4.4 Summary

In this chapter, we have shown to extract multiscale features from large-scale point clouds using averaged features and optimal scale features and perform semantic classification using these features in the Apache-Cassandra integrated framework. We have defined the new approach to perform the customized region-based partitioning using Spark custom partitioner and integrating it with Cassandra to ensure data locality. We

have included the buffer regions to ensure no loss of neighbor information and zero inter-node communication latency. Our use of cubical neighborhood to generate the local descriptor reduces the computation complexity while generating the features. We also show that by removing the buffer region, we save **16%** storage by just compromising **2%** in overall accuracy. The overall architecture is horizontally scalable. We have been able to show the efficacy of our system for 0.5 billion points, with a scope of handling bigger data and more scales in future work.

CHAPTER 5

ADAPTIVE MULTISCALE FEATURE EXTRACTION

Multiple spatial scales have been used extensively for feature extraction from LiDAR point clouds. These features have been used for semantic classification, segmentation, and other data analysis methods. There is a gap in the adaptive methodology for the effective use of multiple scales here. This stems from determining the best strategy to aggregate the information or features gathered from different scales. The widely used multiscale method is feature extraction at an optimal scale, which is in itself an adaptive method. However, the success of identifying the optimal scale depends on the set of scales used in its determination, as it must include the scale where the global minimum of eigenentropy occurs. An alternative method is to average features across multiple scales, which works in specific scenarios. In order to improve the flexibility of using different methods in the same workflow, we propose an adaptive method for the selection of multiscale feature extraction for semantic classification of LiDAR point clouds, with a focus on airborne laser scans. Our decision-making process for finding the best multiscale method exploits the spatial locality of the features. We show how such a control strategy can be implemented in an Apache Spark–Cassandra distributed system for processing large-scale point clouds using voxelization for preserving spatial locality, and binomial logistic regression for selecting voxels to implement a specific multiscale method at. Our results show significant improvement in classification accuracy in the Dayton Annotated Laser Earth Scan (DALES) data, implemented using

Spark MLlib in our distributed system.

Many of the features that are significant for classification are derived from the local neighborhood. Use of multiple spatial scales for gathering salient information has been practiced for geometric analysis of LiDAR point clouds to characterize spatial locality effectively [6]. There are different ways of aggregating the information across scales, which influence the value of the extracted features. Using a scale where global minimum of entropy occurs as optimal and subsequently for feature extraction [6] has been widely practiced for LiDAR point cloud classification [12]. However, this method is fraught with the appropriate choice of initial scales which must contain the scale with global minimum value of entropy. As an alternative to optimal scale determination, average of relevant features across scales has been used for geometric reconstruction and semantic classification has been explored to a limited extent for airborne LiDAR point clouds [29]. However, averaging works only if there is limited (statistical) variation in the feature values across different scales. Yet another way of aggregating scales is to use the features from all scales as a long vector [30], which requires neighborhood approximation with high point density. The state-of-the-art implementation of long vector is significantly different from those of optimal scale and averaging approaches. Optimal scale as well as scale-averaged features have shown good results in semantic classification of airborne LiDAR point clouds [12,29]. Given the limitations of both methods, an adaptive choice of the multiscale feature extraction at each point between the two methods is bound to improve the classification accuracy, to be tested on a selected classifier.

The point-wise computation of multiscale features using is in itself a compute-intensive problem. The compute requirements become exacerbated for an adaptive multiscale method for large-scale point clouds. Hence, we propose a control strategy of reuse of computed features that leads to an efficient implementation of the adaptive method on our proposed distributed system (Chapter 3).

Multiscale methods exploit local dependencies. In order to exploit the spatial locality in the adaptive multiscale method, we propose voxelization of the point cloud. We then use these voxels as the smallest units to determine the *local* strategy for choice of multiscale method. We also choose voxelization to be a part of the implementation on the distributed system, as a process *integratable* with the spatial partitioning and the parallelism in the system.

Ground truth is an ideal metric to make the decision between the two methods at each point. However, in a real-world implementation, the ground truth is usually unavailable. Hence, we propose a novel approach where the voxels are classified based on its better performance in the semantic classification. Posing this as a binary classification problem, we propose the use of binomial logistic regression model (LRM). We choose random forest classifier (RFC) for semantic classification owing to its suitability for Airborne Laser Scanning (ALS) point cloud analysis [12] and the availability of its implementation in Spark MLlib.

Adaptive strategies in analyzing airborne LiDAR point cloud data have been studied in the recent past. A collection of spherical and cylindrical neighborhood shapes, including optimal scale for the former, has been used for feature extraction [31]. These features after normalization have been used in an RFC. The classification results show improvement in overall accuracy, when using the adaptive multiscale features. We use cubical neighborhood in our distributed system owing to the trade-off between computation and accuracy, and its approximation to spherical neighborhood [1]. Since computing features using different neighborhood shapes is not an efficient strategy for large-scale point cloud processing, we explore approaches where the same features can be reused in different ways. This is the overall motivation behind our adaptive strategy. In this chapter, we explain our contributions in adaptive multiscale feature extraction for semantic classification, which include:

- a novel adaptive control strategy of combining multiscale feature extraction methods, namely, optimal scale and averaging operation, to improve accuracy of semantic classification of ALS point clouds,
- an effective application of voxelization for preserving spatial locality of implementation of selected multiscale method,
- a novel use of logistic regression to predict the classification outcome using widely used optimal scale approach in the random forest classifier, in order to select the voxels for changing to the multiscale averaging method.

5.1 Adaptive Multiscale Feature Extraction

We propose a control strategy of adaptive multiscale feature extraction using optimal scale and averaging operation for improving accuracy of semantic classification of ALS point clouds. For assessing our proposed strategy, we use ground truth to confirm our hypothesis. We then propose an efficient data management strategy to implement multiscale aggregation using point cloud voxelization and logistic regression.

5.1.1 Control Strategy – Rationale and Implementation

While theoretically, the global minimum of entropy measure is an effective indicator of optimal scale [6], the global minimum may not always be present in the selected scales, in practice. This leads to the case where not all regions in the point cloud are best represented using optimal scale. In practice, determining optimal scale requires an exhaustive search for the global minimum entropy across multiple scales. Especially in large-scale point clouds, a trade-off between efficiency and accuracy is used, where a local minimum is often chosen.

As a result, we hypothesize that regions with low classification accuracy owing to this trade-off, can improve the accuracy using averaged features. We use the ground truth to test this hypothesis. Ground truth identifies points misclassified using the optimal scale approach, but correctly classified using the averaging approach. The accuracy is then improved adaptively by replacing the optimal scale features with averaged ones.

Definition: The *adaptive multiscale feature extraction* is the control strategy of choosing the best multiscale features at each point in a point cloud to improve the classification outcome.

5.1.2 Voxelization

While we have incorporated spatial locality in local geometric descriptors, we also expect that the spatial contiguity (locality) influences the multiscale approach for the region. Using spatial locality for decision-making can be achieved using our distributed system, designed for managing and processing large-scale point clouds. We have shown in our previous work that an Apache Spark-Cassandra integrated distributed system can be effectively used for semantic classification of large-scale airborne LiDAR point clouds, using both averaged multiscale features [1] as well as optimal scale ones [2]. The spatial partitioning used in the distributed system is along either x- or y-axis. This partitioning is used for distributing the data to different Spark nodes. For region-wise locality, we now voxelize each spatial partition along both x- and y- axes in a gridded format, thus forming $v_x \times v_y$ cuboidal voxels. Now, we propose to apply a chosen multiscale approach to all points in a voxel, *i.e.,* a subset of the point cloud, thus implementing a region-wise application of a chosen method. For the sake of simplicity, we keep $v_x = v_y$ here. The voxelization of the point cloud also improves system efficiency through increased parallelism, as the voxelization and regional analysis are implemented in the same Spark node.

5.1.3 Voxel Selection Using Logistic Regression

We first rank voxels based on their average IOU values when classified using the optimal scale features. We select the low-ranked voxels, whose points are then classified using the averaged features.

We use ground truth analysis to establish the improvement in classification accuracy. However, in practice, an automated method is needed for selecting voxels for applying the change in the multiscale method, *i.e.,* in the absence of ground truth. Given that there is no single feature that gives clear clusters for LiDAR point cloud classification, we need a combination of features in the feature vector that can *predict* the classification outcome for a voxel. We now pose this problem as that of binary classification, given the voxels are to be grouped based on its apt multiscale aggregation method, from our two selected methods. Hence, a binomial logistic regression model (LRM) is applicable where the outcome (dependent) binary variable is the success of classification using our chosen classifier. We use the following point-wise feature vector for the logistic regression, which is derived from the same used in the classifier:

$$\mathrm{F}_{LRM}=\mathrm{F}_s \setminus \{C_s, \Delta_z\} \quad .$$

We remove C_s and Δ_z from the feature vector, as they are linearly dependent on the remaining saliency map, $\{C_l, C_p\}$, and height z, respectively. The choice between z and Δ_z has been decided based on our results. The LRM is computed and applied point-wise.

Since the number of points in each voxel is not a constant, the feature size varies at the voxel level. Hence, performing logistic regression directly for voxel classification based on a long vector of point-wise features is inefficient and impractical. Hence, we perform point-wise analysis using logistic regression, and then we determine the *success rate* for each voxel. We use a threshold for the success rate τ_s, lower than which, we select the voxel to change its multiscale method for feature extraction. We expect the voxel size and voxel-wise point density to influence the improvement in

accuracy. We study the influence of voxel size in our results, here. The voxels could be further merged or split to achieve a desired point density in each voxel for effective implementation. This leads to adaptive voxel sizes, which is to be studied in future.

5.2 Workflow

In our proposed workflow (Figure FC5.1), we first train the RFC models for both multiscale methods, namely, RFC-1 and RFC-2 for the optimal scale and averaged features, respectively. We then use the accuracy of RFC-1 outcomes to train the binomial LRM, now using *normalized* features. For testing an unlabeled point cloud, we first run the RFC-1, to get the point-wise class labels. We run the voxelizer and the LRM, after storing the class labels from the RFC-1. If the LRM gives outcome '1' for a voxel, *i.e.,* success in running RFC-1 for the voxel is above the threshold τ_s, then we retain the labels from RFC-1 for all the points in the voxel. If the outcome for the voxel is '0', then we compute averaged features for only those points and implement RFC-2 on the voxels. The class labels of all points in the voxel are now the outcomes of RFC-2. Our workflow finally outputs labeled point clouds.

An alternate to the strategy of *correcting* optimal scale features using averaged features, is to *choose* between the two approaches. However, choosing between two approaches requires the implementation of both approaches for each point, which is inefficient for large-scale point clouds. This step is unavoidable for training but can be avoided for testing, and using pre-trained models for testing can keep it as a one-time process. Thus, we adopt a sequential implementation of one method, followed by correction of selected regions using the second method, which is more efficient for big data.

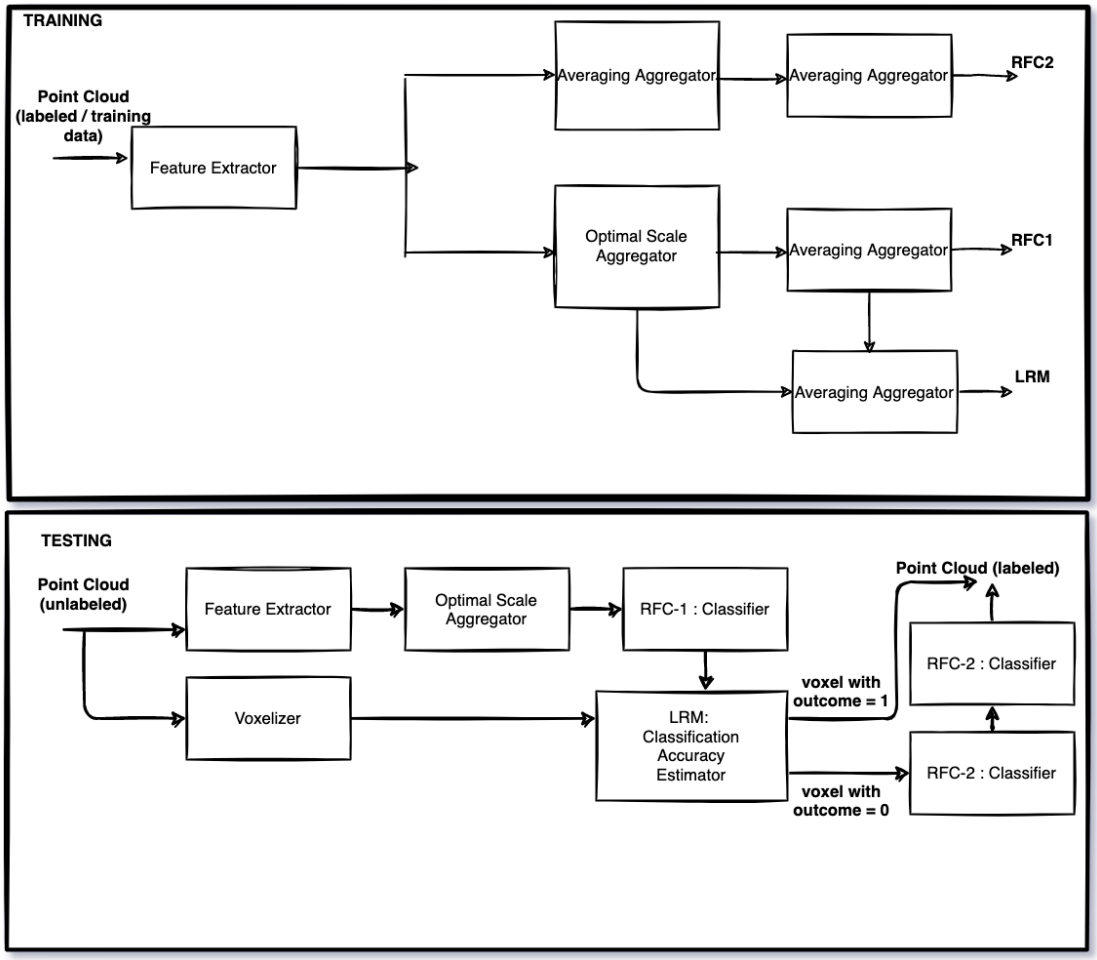

Figure FC5.1: Our proposed workflow involving our adaptive strategy of multiscale aggregation and a logistic regression model for improving classification accuracy in large-scale point clouds using an Apache Spark-Cassandra integrated distributed system [1, 2]. Models RFC-1, RFC-2, and LRM are trained using Apache Spark MLlib.

5.3 Experiments, Results And Discussion

For our experiments, we have used the DALES dataset [5]. We split the data for training and testing for the Random Forest Classifier (RFC) in the Spark MLlib (Figure FC5.2 and Table TC5.1). We have used the distributed system with the specifications given in Section 3.4.1.

We split the testing data used for the RFC further, as training and testing the binary

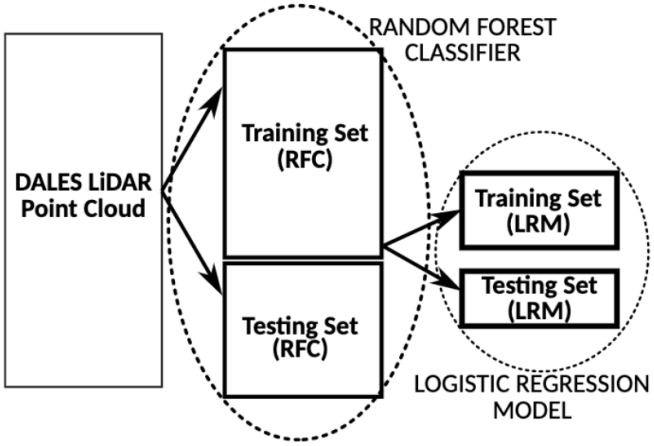

Figure FC5.2: Our implementation of splitting data for training and testing in the Random Forest Classifier and Logistic Regression modules in Spark ML, in our Apache Spark-Cassandra integrated distributed system.

logistic regression model (LRM), also implemented using Spark MLlib. We have used four tiles for training the RFC, and two tiles for RFC testing, which are used again for LRM training and testing (Table TC5.1). We have used a $\sim$70/30 split for training and testing on the RFC. For multiscale feature extraction, we have used ten uniformly distributed scales with cubical neighborhood sizes l=1m, at minimum, l=10m at maximum, and Δl=1m as the scale increment.

In terms of performance, our distributed system takes 2.287s for per-point processing, followed by 117μs for per-voxel processing. Table TC5.2 gives results for both point- and voxel-wise classification. We observe from the point-wise results that the averaging approach by itself significantly outperforms the optimal scale approach. Hence, we use a conservative success rate threshold τ_s= 90% here. We first explore our hypothesis of the implementation of adaptive strategy of multiscale averaging and voxelization improving the classification accuracy. The accuracy at point-level using ground truth analysis is the baseline (Table TC5.2). We see that, as the voxel size reduces the accuracy improves, but at the same time, the point density is influenced by the voxel size. At 320$\times$320 voxelization, we have an average of $\sim$900 points per voxel, that describes

Table TC5.1: Tiles in DALES dataset [5] used in Random Forest Classifier (RFC) and Logistic Regression Model (LRM)

Tile ID	#points	Tile ID	#points	+Roles (LRM)
RFC Training		RFC Testing		
5110_54320	17,747,769	5150_54325	11,882,667	Train
5110_54460	13,784,200			
5110_54475	11,981,458	5080_54400	12,219,779	Test
5110_54495	11,930,713			
Total: 55,444,140 points		Total: 24,102,446 points		

Table TC5.2: Intersection Over Union (IOU) and Overall Accuracy (OA) measures of Semantic Classification by RFC

Granularity Level	Average IOU	OA
Point-wise	Using Optimal Scale	
	0.2782	74.18%
	Using Multiscale Averaging	
	0.3312	85.48%
Proposed Adaptive Multiscale Feature Extraction *(Voxelization along x- and y-axes gives $v_x \times v_y$ voxels)*		
Point/Voxel Selection based on Ground-truth Analysis		
Point-wise	0.3333	85.611%
Voxel-wise		
10×10 voxels (50.00m×50.00m)	0.3380	85.49%
20×20 voxels (25.00m×25.00m)	0.3395	85.44%
40×40 voxels (12.50m×12.50m)	0.3452	85.53%
80×80 voxels (6.25m×6.25m)	0.3476	85.62%
160×160 voxels (3.12m×3.12m)	0.3514	85.77%
320×320 voxels (1.56m×1.56m)	0.3520	85.79%
Voxel Selection based on Logistic Regression Model		
Voxel-wise		
10×10 voxels (50.00m×50.00m)	0.3378	85.22%
20×20 voxels (25.00m×25.00m)	0.3395	85.61%
40×40 voxels (12.50m×12.50m)	0.3381	85.59%
80×80 voxels (6.25m×6.25m)	0.3384	85.68%
160×160 voxels (3.12m×3.12m)	0.3391	85.60%
320×320 voxels (1.56m×1.56m)	0.3520	85.72%

The metric size of each voxel is given in parantheses in the leftmost column.

spatial locality sufficiently. We expect that any further increase in the number of voxels will not improve results owing to the reduction in point density.

We observe that running LRM in the voxelized test data, at different voxel sizes, gives comparable results to that of the ground truth analysis, thus demonstrating that our choice of logistic regression model is effective. We have repeated the experiment after swapping the LRM training and testing tiles (Table TC5.1). However, this swap deteriorates classification to 73.45% OA with 0.3032 IOU for optimal scale, and 74.84% OA with 0.2825 IOU for averaging. We observe that the tile `5080_54400` has relatively more class imbalance, in comparison to the tile `5150_54325`. Overall, the class balance of the tile for LRM training is a determinant of the success of our proposed method.

We have also repeated these experiments for the Vaihingen benchmark dataset [32], for proof of concept. The dataset has an average point density of 5-7 ppm, with total of 1.2 billion points. The experiments could not be run for more than 20×20 voxelization owing to the high sparsification of the voxels. The voxel selection using ground truth analysis gives 69.02% OA and 0.2059 IOU; and using LRM gives 68.73% OA with 0.2078 IOU, when using Area-3 (323,895 points) as training and Area-2 (266,674 points) as testing. We have used three scales with sizes l={3.78m, 4.20m, 4.62m}, based on our previous work [29]. We observe and conclude that the high point density is indeed a determinant of the success of our proposed method.

On comparing the RFC results in Chapter 4, we can now conclude that by deploying adaptive method we could able to improve the **OA** from **85.48% to 85.79%** and **IOU** from **0.3312 to 0.3520**.

5.4 Summary

Our proposed workflow has revealed salient observations on the dependency of the voxel size on the classification accuracy, when using our control strategy and adaptive multiscale method. Further analysis of the voxelization for automated decision-making using logistic regression is required.

Our work opens up novel adaptive methods of extracting hand-crafted features to be used in different learning methodologies, namely, unsupervised and supervised learning. Recent studies have shown that appropriate hand-crafted features used in random forest classifier can define the convolution function used in more recent deep learning methods [33] for segmentation and classification of 3D point clouds.

CHAPTER 6

INTERACTIVE VISUALIZATION SYSTEM

One of the data analysis processes to understand 3D LiDAR point clouds is to perform visualization to allow data exploration for further analysis or annotation. In conventional visualization approaches, the points in the point cloud are either considered raw or are converted into triangle models or images, thus providing for rendering using different visualization methods. Our goal is to build a visualization system that supports not just point cloud visualization in its raw format, but also the intermediate and final outcomes of data analytics, namely, extracted features and semantic classes.

The traditional approach to visualize requires transferring data to stand-alone devices and install a visualization tool. Additionally, this requires specific configurations to be present on the device, such as CPU, RAM, or GPU. As the data size increases, loading and transferring the data from one device to another becomes challenging. For example, to share any analysis report for the larger audience for collaborative work requires sharing the data and loading it in the target device. For large-scale data sizes of the order of gigabytes or terabytes, the whole process becomes infeasible and costly.

With the advances in OpenGL (Open Graphics Library) extensions to web browsers, namely, WebGL, 3D visualization, and user interactions have become available on standard browsers and natively supported by all browser engines, and even on mobile devices. This has facilitated the distribution of 3D content on the browser quickly without

the need for a particular device. Also, with the advancement in web technologies to load data asynchronously and process using the worker threads allows the user to keep the web browser application interactive while doing all the heavy lifting in the background in parallel.

For smaller datasets that can fit in the device memory where the browser is running, the data loading and visualization are straightforward, without any special memory management. But, as the data size increases, the data may not fit in the memory, and the time to download may range from minutes to hours, thus affecting the interactivity. Thus, visualization becomes a challenge for large-scale point cloud datasets. Potree [7] has adopted the point cloud rendering on limited resource devices by supporting visualization on standard web browsers to overcome these limitations using the modifiable nested octree (MNO) structure.

Different from Potree, we propose the use of our distributed system to provide real-time visualization of analytic processes, *e.g.,* feature extraction and semantic classification. We propose providing the interactions to perform real-time operations and analytics on a large-scale point cloud. This is possible by using system integration with Cassandra for the data management for point cloud data instead of particular files or data structures. Also, it is the extension of our integrated Apache Spark-Cassandra architecture described in Chapter 3. Our proposed architecture of a visualization system that is peripheral to the integrated distributed system is as shown in Figure FC6.1, we can see the working system is divided into three components, namely, the browser-based visualization tool, the Spark-Cassandra cluster for processing and managing the data, and finally a set of services between the tool and the cluster. The services include scheduling the jobs, loading point clouds, and performing analytics. The browser-based visualization tool has in-built features to allow interaction and dispatch requests to our services to perform various tasks. It helps us perform real-time operations like local geometric descriptor generation, classification using pre-built classifier models, analytics

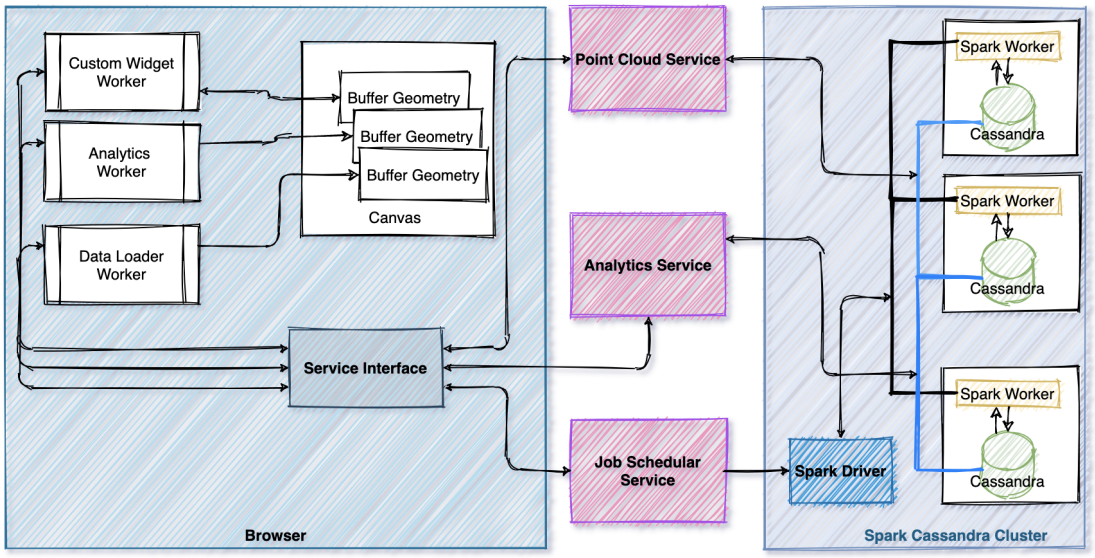

Figure FC6.1: Our proposed architecture for an interactive visualization system peripheral to the Apache Spark-Cassandra integrated system.

(*e.g.,* class distribution), and point cloud visualization on our browser-based visualization tool/dashboard.

A known limitation of our browser-based visualization tool is that it is not optimized for the high-quality graphical rendering of the point cloud, as is done in Potree [7]. This is owing to our current goal in providing a working system that provides interactive visualization of the point cloud by communication with our distributed system, so that we can also perform real-time processing and analytics using our Apache Spark-Cassandra cluster.

6.1 Distributed Data Structure

The raw point cloud gets partitioned along the principal axis (X or Y) based on maximum range (Section 4.1.2), and then the entire region is partitioned into multiple regions. The partitions are then distributed across the Cassandra node, and each node stores a region. The primary key used to store each row consist of (`RegionID`, `X`, `Y`),

where `RegionID` is the partition key, and `X,Y` are clustering keys sorted in order. This is how the raw point cloud is stored. In addition, any intermediate data like local geometric descriptor, feature vector, and classification results also get stored using the `RegionID` as the partition key and `X, Y` as the clustering key. This method of storing data allows querying Cassandra most optimally to retrieve the data inside a voxel bounded by `X,Y`.

6.1.1 Subsampling

The original point cloud data can be very dense and large-scale, in our application. Loading and visualizing these points in the browser for a larger area will require large memory footprint, and transferring the data incurs network latency. In addition, rendering and frame refresh on a browser will become slow, impacting the interactivity of the visualization application. The solution is to get an optimal sampling of the original point cloud data to preserve the underlying (geometric) surface model. Our requirement also includes that the number of points in the sampling must be limited by the maximum that can be rendered on the targeted device browser.

Voxel-grid subsampling, as shown in Figure FC6.2, approximates the set of points inside a voxel with the centroid of these points. While approximation using the voxel center is faster than using the centroid, but the irregularities inherent in the point cloud are best represented using the centroid. Thus, the centroid of points in a voxel approximates the set of points in the voxel and its underlying surface more accurately.

The point cloud is spatially segmented into smaller cubes or voxels. The size of the voxel determines the sample size and the amount of information that is retained from the original data. The smaller the voxels, the more information is retained, but the size of the sampling itself increases.

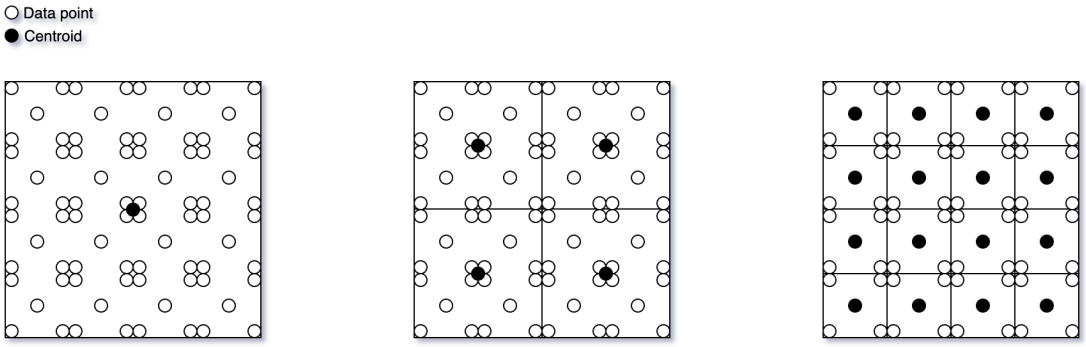

Figure FC6.2: An example demonstrating the change in voxel grid sub-sampling with change in voxel sizes.

6.1.1.1 Implementation

We load the already partitioned point cloud into multiple regions in Spark as RDD, where each region also contains the left and right buffer points as a neighbor. Then, we identify all the voxels inside the region with the chosen voxel size. And replace them with the centroid of the points inside the voxel. The resultant RDD then gets persisted into the Cassandra for visualization as subsampled data. Additionally, we can define the voxel sizes for different resolutions and persist the sub-sampled points for visualization on demand. In this thesis, we limit our implementation to voxel size of (0.5m, 0.5m, 0.5m). Experimenting with different voxel sizes may be considered for future work.

We store the subsampled data for each region in cache under the *Point Cloud Service* using the cache aside strategy. The use of cache allows efficient loading of the subsampled data on the browser over the network. Thus, this reduces the redundant read calls to Cassandra.

6.1.1.2 Cassandra Query to Optimally Read

Cassandra Schema is build on the queries, unlike in the relational databases. Following the best practice in using Cassandra through even distribution of data in the cluster,

we partition the data along the principal axis (X or Y), with maximum range of values, as explained in Section 4.1.2.

Also, to avoid reading the data from multiple partitions against select query, we restrict our any query to any specific region or partition at a time. To load the data, we execute the query to load the points in batches in multiple pages to maximize the throughput and minimize network latency. To define multiple page searches, we now divide the principal axis in the region into multiple subintervals, and for each subinterval, the range search loads one page.

```
SELECT * FROM gvcl.PointCloud
WHERE RegionID = 1 AND (X,Y) >= (1.5, 3.0) AND (X,Y) <= (3.0,
4)
LIMIT 100000;
```

We choose the voxel-grid based sampling instead of the Poisson disk sampling used in Potree [7] in order to reduce the number of computations. Our overarching objective of building a visualization tool peripheral to our distributed system entails intensive computations and communication for user interactivity, and analytics. The voxel-grid sampling strategy is an optimal trade-off between computational complexity and accuracy. Improve the quality of graphical rendering for visualization and integrating features available in Potree for efficient visualization are in the scope of our future work, while using Cassandra instead of filesystem for persistent storage.

6.2 Rendering on the Web Browser

The cached sub-sampled points are read from the Cassandra to visualize on the browser using WebGL. The point rendering using the interactive 2D and 3D graphics

on the browser has become possible with the introduction of WebGL support, thus without having to use plug-ins and HTML5 canvas.

6.2.1 WebGL

WebGL is an interactive 2D and 3D graphics library written in javascript that provides API, which confirms OpenGL ES 2.0 (for embedded systems) and can be rendered in HTML5 canvas. The API uses the hardware graphics acceleration provided by the device, thus, making the application GPU-accelerated. WebGL is a low-level library, that can be thought of as a rasterization engine. WebGL facilitates rendering of points, lines, and triangles, only. To accomplish a task and write code to achieve an average result in WebGL requires expertise. Three.js is an alternative library to direct WebGL programming, where Three.js library is a layer above WebGL. Thus, Three.js has simpler-to-use wrapper methods for generating scenes, materials, textures, etc., and performing 3D mathematical operations.

We used Three.js here for rendering the point clouds, using the concept of BufferGeometry provided in the library. BufferGeometry represents mesh, line, or point geometry. It involves multiple buffers, where each buffer is dedicated to different attributes, namely, vertex positions, face indices, normals, colors, UVs, and custom attributes. Thus, BufferGeomtry reduces the cost of passing this data to GPU for rendering. To create these buffers, we use BufferAttributes in Three.js. These memory buffers act like a fixed buffer and can be reused to update the attribute values selectively. We used fixed buffers size by setting *point budget*. We use BufferGeomtery to update selected attributes interactively for specific visualizations.

6.2.2 Asynchronous and Parallel Processing

Javascript is a single-threaded program, and multiple scripts cannot run at the same time. We essentially have two types of tasks, IO intensives like loading data from service and CPU intensive, like processing some maths on the data. Hence, if we run any time-consuming computation task, our thread gets blocked, and UI becomes unresponsive, and we have to wait until the task gets completed to become responsive.

Its callback method allows running time-intensive tasks without blocking the program under the same thread. This allows us to build interactive GUI, without blocking any other processes during resource-intensive background processes, *e.g.,* loading data, updating buffer, etc. The non-blocking strategy also allows access to the updated frame buffer as soon as they are ready. This works well especially when IO-bound tasks may block the thread of CPU-bound tasks.

In comparison, parallel processing on GPU requires creating multiple threads, dividing our tasks into smaller parts, and executing them in parallel with load balancing across all threads. The *web workers* on the browser runs parallel tasks using the background threads. We deploy the web workers for the CPU-bounded tasks in our application.

We use asynchronous methods for most of our web API interactions and for dispatching the tasks to the web workers. We use a parallelized approach to generate analytics (*e.g.,* class distribution), data processing, or performing any mathematical operations on the data. For instance, we use web workers to transform the data into objects required by the graph API for plotting the analytics in the visualization.

6.3 Back-end Services

We build the set of back-end services to expose the APIs to interact with our Apache Spark-Cassandra cluster, which processes and manages the point cloud data. These services orchestrate the task that can be performed. These tasks are mostly IO-driven to access the data or schedule the task. The services also include getting the status of these tasks.

6.3.1 Point Cloud Service

The Point Cloud service exposes the set of APIs to manage the point cloud data and its intermediate processed data stored in the Cassandra cluster. It directly interacts with the Cassandra cluster to retrieve data. We create the task profile for each task and maintain its meta-information like Cassandra Table Schema information, point cloud metadata, and tasks information. Our services are written in Node.js using Typescript. These services expose the following set of APIs:

- To upload a point cloud file or configure the data source. Currently, we can configure S3 as the data source. Support for Hive or any other supported file system can be given in the future.
- To create/update and manage the task-related information like table schemas, point cloud metadata, start-up tasks, and other tasks to perform on point cloud data. Table schemas are defined to store raw and processed point cloud data like local geometric descriptors, feature vectors, classification results.
- To setup the cluster for start-up task defined in the previous API. It creates tables and validates, if the required resources are there to start the process.
- To retrieve data from various tables on demand by executing the Cassandra query.

6.3.2 Analytics Service

We need to build an aggregated report or expose the analytics report generated on a particular analysis done on the point cloud directly or on the processed data. The service is also responsible for computing some lightweight analytics reports like class distribution reports on data requested to generate the view-ready report. Such customized APIs can be built in this service to generate reports that can be done here. We build two APIs for our use case, the class distribution report and the local geometric descriptors distribution report for each semantic class.

6.3.3 Job Scheduler

The Spark-Cassandra cluster can take only a limited number of requests at a time, depending on the cluster configuration. This requires us to build a service that can schedule the submitted task for processing and manage the status of the task. It exposes two APIs – one to submit the task already created using the aforementioned Point Cloud service, and the other to publish the status and other information related to the progress of the scheduled task.

6.4 Tools and Interactions

The dashboard, shown in Figure FC6.3, shows our proof-of-concept of an interactive visualization tool for the point cloud and its analytics related to semantic classification. Generating the real-time analytics pertains to the semantic classification of the point clouds using the pre-build ML models, as discussed in Chapter 3.

The current version of our tool provides the building blocks that can be enhanced in future work. Here, our objective is to show the possibility of rendering large-scale

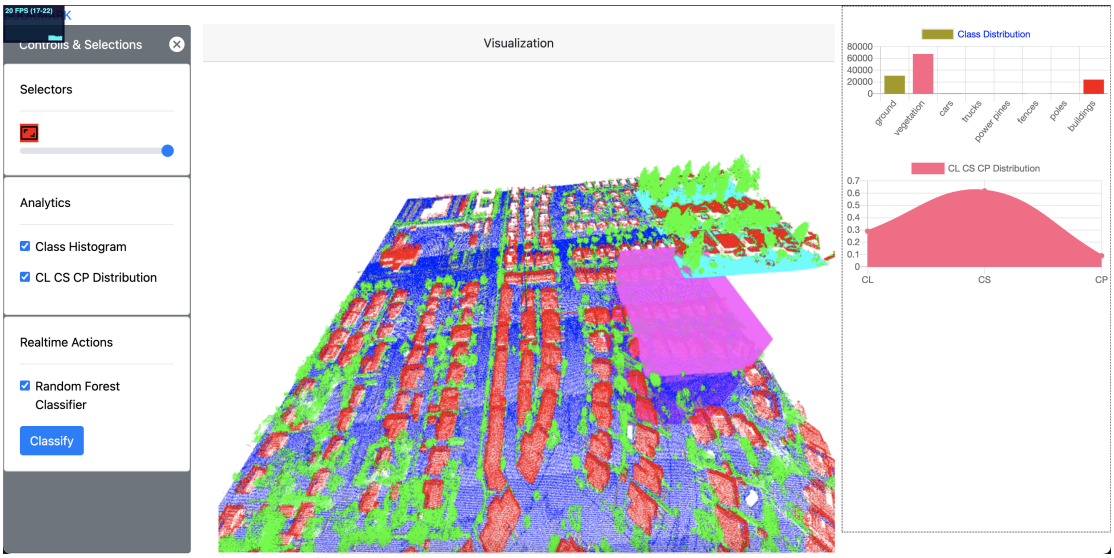

Figure FC6.3: Our proof-of-concept of a dashboard of point cloud processing and analytics running on a web browser.

point cloud, interaction, and associated analytics using our distributed system.

The right tab of the tool includes the analytics widgets. The tool exposes the set of APIs to build new analytics widgets to be integrated into the tool. These widgets use the web worker to generate the computations required for the analytics report. The tool uses a *selection* box, which allows the user to interactively select a region in the point cloud using a bounding box. The selection box is a magenta box that the user can translate and scale, and the points contained in the box are the selected points. Now, the widget receives the selected point cloud data and the selection box information as the input message to the worker, which can then be used later to dispatch any request to our back-end services.

The left tab has widgets to perform actions, such as selecting point cloud regions (using the selection box), enabling visualization of analytics, and scheduling the classification task using the pre-build ML models. While the classification progresses, we fetch the results from our back-end service and visualize the progressive results in real-time by updating the geometry buffers.

More on the Selection Box: The selection box allows the user to select the region of interest (ROI) in the point cloud by highlighting the points inside the selection box, as shown in Figure FC6.3. The intended use case of the selection box is to render the analytics of the points in the ROI, or to run any new task like classification using classifier specifically in the ROI as an exploratory task. The box position is set at (0,0,0) and the slider is given to set the size of the box to perform the selection.

6.4.1 Navigation Controls

Single navigation may not be able to fulfill all the desired interactivity. We choose OrbitControls as the navigation tool. As the name suggests, it orbits around a target or pivot. The rotation is constrained along the Y-axis by locking it in the positive up direction. It prevents any titling off the axis and allows the user to orbit, zoom and pan using the mouse buttons.

6.4.2 Point Budget

We use the buffer geometry for our rendering. The user is given the option to fix the size of the points that can be efficiently handled by the browser for rendering and interactions. We cannot resize the buffer geometry, as it is very costly and equivalent to generating a new geometry. However, we can always update the buffers. It puts the restrictions on the size of the buffer for an attribute and pre-allocates the size which can cater the future needs as well. This also implies that we cannot create a buffer geometry that can extend infinitely. With these restrictions, we set the maximum point budget to 10 million. The user can always set this budget as per the requirement of the input data.

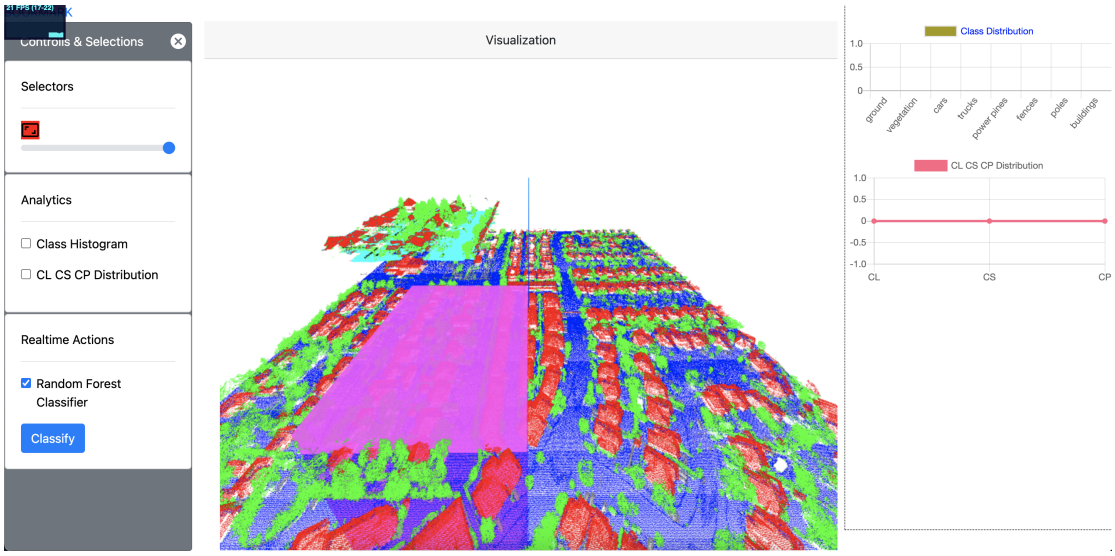

Figure FC6.4: A demonstration on our browser-based visualization tool, of real-time visualization of progressive semantic classification on a selection of points, indicated by the magenta box, and the updated selection is shown at a height from the point cloud to observe the streaming changes in semantic labels.

6.4.3 User Interface

Our tool provides the control to interact with the point cloud and gives options to perform actions on the selected region of interest. It also exposes the method to build custom actions and interactions using the widget framework. In comparison, Potree is a similar tool [7] provides a user interface to perform measurement, clipping and annotating the points in point cloud, but does not provide any facility for performing classification by design. On the other hand, our tool provides the capability for performing semantic classification of airborne LiDAR point clouds, including real-time analytics and classification using pre-build models on the region of interest.

6.4.3.1 Classifier Selector And Real-time Classification Visualization

The classifier selector tool provides the list of pre-build classifiers for the point cloud loaded as shown in Figure FC6.4. On clicking the tool, the classification job gets

submitted to the job scheduler service, which internally schedules the classification job on the Spark-Cassandra cluster. Once the classification job starts on the cluster, the results also start getting stored in the Cassandra cluster. Our background worker starts pulling the results, as they become available and progressively updates the buffer for progressive visualization of the classification results in real-time.

6.4.3.2 Analytics Widgets

The tool framework exposes a set of methods to create widgets using the message passing by the web workers. Each widget receives the bounding box information of the selection box message in the worker method. Then the worker uses this information to get any analytics report details from the back-end analytics service or get the details from the point cloud service and build the report or data in the worker method. We build a few widgets as a working example.

6.4.3.3 Class Distribution

The class distribution widget shown in Figure FC6.5, shows the histogram of the object classes of the points. The widget receives the selection box bounding information and dispatches a request to the analytics service, pulling the point from the Cassandra cluster. Using the meta-information, we check the point regions, prepare the query for each region, and read the required data. Once the data is received in the analytics service, it builds the class count report, and the response is sent back. Once the widget receives the response, it sends the message to the UI for visualization in the charts.

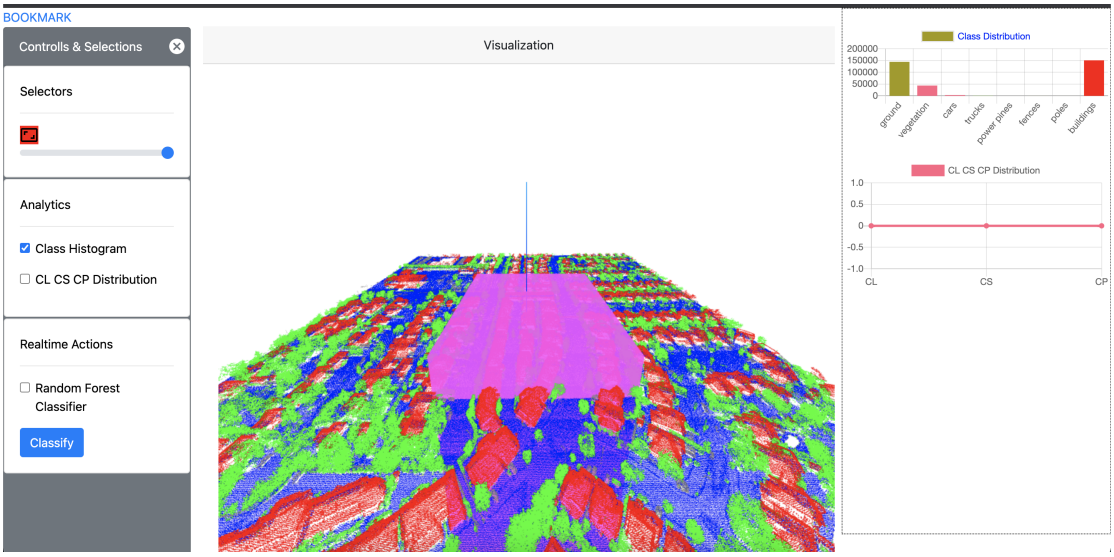

Figure FC6.5: Real-time update of analytics on the selected region in point cloud (shown in magenta) demonstrating the class distribution histogram on our browser-based visualization tool.

6.5 Results

The performance of the browser tool is measured in frames per second (FPS), and the rendering time is measured in milliseconds against the point budget. In our visualization system, the GPU decides the rendering performance, and hence, our application has a little contribution from the CPU.

For interactive GUI applications, 60fps is considered as the best, and 30fps is acceptable real-time performance, and the interval 10-30fps is considered to be interactive. When the frame rate is below 10fps, then either the results should be discarded, or the point budget should be reduced.

We have tested our application on a MacBook Pro with, 2.3 GHz Dual-Core Intel Core i5 CPU, 8 GB 2133 MHz LPDDR3 Memory, and GPU Intel Iris Plus Graphics 640. Our results show the effectiveness of the tool and achieve real-time or interactive rendering performance. We observe that the fps rate decreases with the increase in point budgets as shown in Figure FC6.6. We also observe that the rendering time shows

a linear relationship with the size of point budgets as shown in Figure FC6.7.

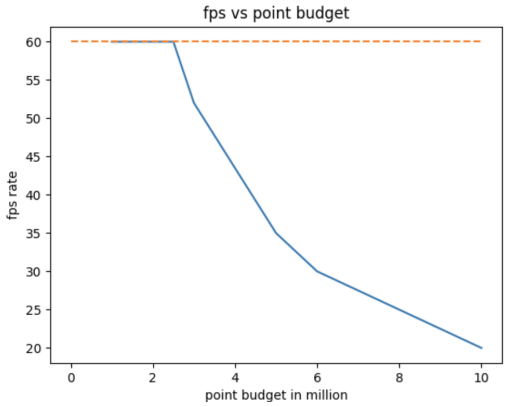

Figure FC6.6: Frames per second for the DALES data set with different point budgets, as used in the buffer geometry for our visualization tool developed using WebGL.

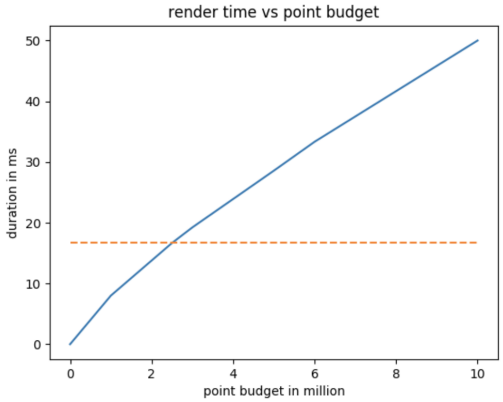

Figure FC6.7: Performance plot showing the rendering time taken by the point cloud with different point budgets in our WebGL-based tool.

6.6 Summary

We have demonstrated the visualization tool that can render and visualize large-scale point clouds on a web-browser, by partitioning the data along the principal axis, subsampling, and storing in Cassandra. Splitting the data into smaller regions and using the voxel-grid subsampling of points allows to load the reduced point cloud for rendering. Our distributed approach reduces the time required to generate the voxel-grid sampling in large-scale point clouds. The caching of the partitioned sub-sampled data at the server, and the loading of the point cloud data for each region implemented in parallel from the cache reduces the time to load the data on the browser. The interactions feature to perform real-time analytics and scheduling processing tasks like classification of selected regions using the pre-built classifier are novel features for any point cloud tool, that provide interactive data exploration and contextual understanding.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

We have presented the system architecture using big data tools to process, manage, and visualize a large-scale point cloud by partitioning the data along the principal axis (X or Y) using the custom partitioning, built on top of an Apache Spark custom partitioner. We have shown the impact of retaining the left and right buffer region in each partition to avoid losing any neighbor information. It became necessary to parallelly process each partition and make the complete strategy horizontally scalable. Later, we have also established that removing the buffer region saves 16% storage by compromising 2% in overall accuracy.

Our proposed methodology has allowed us to generate the local geometric descriptor using cubical neighbor search by approximating the spherical search for multiscale in each partition. We have selectively built feature vectors using raw and local descriptors generated for average and optimal scale. The results generated at each step have been successfully stored in the Cassandra cluster using the same custom partition logic used in the Spark partitioner. We have demonstrated the system IO improvement using the Spark-Cassandra connector to preserve the data locality while executing the task in Spark executor nodes. Implementing the octree or kd-tree on the partitioned data having buffer region and then implementing the k-nearest search can give better neighbor results, which may be implemented in future work. Apache Spark SQL can also

be studied to implement range query search efficiently. Cassandra currently stores X, Y, and Z values under different columns; where Morton code can combine them to one single integer value and reduce the storage and improve the filter query for range search.

We have performed semantic classification on the DALES airborne LiDAR dataset using the average and optimal scale feature vector build using our feature generation methodology. We have then deployed Spark MLlib to build supervised learning models; namely, the random forest classifier and the gradient boosted tree classifier for the semantic classification of points. The accuracy of our semantic classifications is 85% using average scale features and 79% using the optimal scale features. Although we have limited our work to use a few tiles, this work may be extended to all tiles by increasing the cloud computing resources. Performing classification using full training and testing data is desirable as that can improve the classification results.

We have further demonstrated that combining different multiscale methods within a single point cloud yields effective feature vectors to be used in supervised learning. We have designed an adaptive strategy to be implemented on the underline distributed system, building from our previous semantic classification work. Our results show that the use of voxelization to exploit spatial locality of multiscale aggregation and logistic regression to predict the success of random forest classifier improves the accuracy of the classifier. Our proposed strategy demonstrates how extracted features can be used in different ways, namely, multiscale aggregation using optimal scale approach or averaging, to improve classification accuracy. Our adaptive strategy is also generalizable to terrestrial and mobile LiDAR datasets, which needs to be further studied.

In addition, we have implemented the visualization system and built the standard browser-based visualization tool using WebGL to visualize the points in its raw format and its analytics. The framework is integrated seamlessly with our distributed system. Subsequently, we have chosen the voxel-grid subsampling strategy to sample the orig-

inal point cloud. The sampling is implemented on the distributed system in parallel by independently executing the sampling task on each partition. Our current implementation does not consider multi-resolution and frustum views to load more detailed points selectively. Also, other subsampling methods like Poisson disk, random choice, or two-dimensional grid may be tested to improve the build-up times significantly.

We have visualized all the intermediate results like local geometric descriptors, class distribution, and feature vector stored in Cassandra using the analytics widgets on our tool. We have also performed real-time classification by selecting the pre-build classifiers and visualization of the classification results in real-time as the point gets classified. We have built a preliminary set of basic analytics to demonstrate the capability of the proposed framework. More complex analytics can be built using the Spark-Cassandra framework in the scope of future work to visualize the results on the browser.

Our contributions pave the way for novel data science workflows that can be implemented on big data framework for large-scale point cloud analysis. Our work also demonstrates how the big data technology frameworks can be harnessed effectively for solving large-scale point cloud processing problems. Our proposed workflow shows that novel engineering methods of using available features in these frameworks pave the way for more complex data science workflows for spatial data.

Bibliography

[1] S. Singh and J. Sreevalsan-Nair, "A distributed system for multiscale feature extraction and semantic classification of large-scale LiDAR point clouds," in *2020 IEEE India Geoscience and Remote Sensing Symposium (InGARSS)*. IEEE, 2020, pp. 74–77, doi : https://doi.org/10.1109/InGARSS48198.2020.9358938.

[2] S. Singh and J. Sreevalsan-Nair, "A Distributed System for Optimal Scale Feature Extraction and Semantic Classification of Large-Scale Airborne LiDAR Point Clouds," in *International Conference on Distributed Computing and Internet Technology*. Springer, 2021, pp. 280–288, doi : https://doi.org/10.1007/978-3-030-65621-8_18.

[3] S. Singh and J. Sreevalsan-Nair, "Adaptive Multiscale Feature Extraction in a Distributed System for Semantic Classification of Airborne LiDAR Point Clouds," *IEEE Geoscience and Remote Sensing Letters (Early Access)*, July 2021, doi : https://doi.org/10.1109/LGRS.2021.3099935.

[4] J. Sreevalsan-Nair, P. Mohapatra, and S. Singh, "IMGD: Image-based Multiscale Global Descriptors of Airborne LIDAR Point Clouds Used for Comparative Analysis," in *Smart Tools and Apps for Graphics (STAG 2021) - Eurographics Italian Chapter Conference*, P. Frosini, D. Giorgi, S. Melzi, and E. Rodolá, Eds. The Eurographics Association, 2021, pp. 61–72, doi : https://doi.org/10.2312/stag.20211475.

[5] N. Varney, V. K. Asari, and Q. Graehling, "DALES: A Large-scale Aerial LiDAR Data Set for Semantic Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 186–187, doi : https://doi.org/10.1109/CVPRW50498.2020.00101.

[6] J. Demantké, C. Mallet, N. David, and B. Vallet, "Dimensionality based Scale Selection in 3D LiDAR Point Clouds," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 38, no. Part 5, p. W12, 2011, doi : https://doi.org/10.5194/isprsarchives-xxxviii-5-w12-97-2011.

[7] M. Schütz, "Potree: Rendering large point clouds in web browsers," *Technische Universität Wien, Wiedeń*, 2016, url : https://www.cg.tuwien.ac.at/research/publications/2016/SCHUETZ-2016-POT/.

[8] J. Behley, V. Steinhage, and A. B. Cremers, "Efficient Radius Neighbor Search in Three-dimensional Point Clouds," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3625–3630, doi : https://doi.org/10.1109/icra.2015.7139702.

[9] J. Boehm, K. Liu, and C. Alis, "Sideloading-Ingestion of Large Point Clouds into the Apache Spark Big Data Engine," in *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences-ISPRS Archives*, vol. 41. International Society of Photogrammetry and Remote Sensing (ISPRS), 2016, pp. 343–348, doi : https://doi.org/10.5194/isprsarchives-xli-b2-343-2016.

[10] V. Pajić, M. Govedarica, and M. Amović, "Model of Point Cloud Data Management System in Big Data Paradigm," *ISPRS International Journal of Geo-Information*, vol. 7, no. 7, p. 265, 2018, doi : https://doi.org/10.3390/ijgi7070265.

[11] M. Pavlovic, K.-N. Bastian, H. Gildhoff, and A. Ailamaki, "Dictionary Compression in Point Cloud Data Management," *ACM Transactions on Spa-*

tial Algorithms and Systems (TSAS), vol. 5, no. 1, pp. 1–25, 2019, doi : https://doi.org/10.1145/3299770.

[12] M. Weinmann, B. Jutzi, S. Hinz, and C. Mallet, "Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 105, pp. 286–304, 2015, doi : https://doi.org/10.1016/j.isprsjprs.2015.01.016.

[13] P. Keller, O. Kreylos, M. Vanco, M. Hering-Bertram, E. S. Cowgill, L. H. Kellogg, B. Hamann, and H. Hagen, "Extracting and visualizing structural features in environmental point cloud LiDaR data sets," in *Topological Methods in Data Analysis and Visualization*. Springer, 2011, pp. 179–192, doi : https://doi.org/10.1007/978-3-642-15014-2_15.

[14] T. Hackel, J. D. Wegner, N. Savinov, L. Ladicky, K. Schindler, and M. Pollefeys, "Large-scale supervised learning For 3D point cloud labeling: Semantic3d. Net," *Photogrammetric Engineering & Remote Sensing*, vol. 84, no. 5, pp. 297–308, 2018, doi : https://doi.org/10.14358/PERS.84.5.297.

[15] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys, "Semantic3d.net: A new large-scale point cloud classification benchmark," *arXiv preprint arXiv:1704.03847*, 2017, doi : https://doi.org/10.5194/isprs-annals-IV-1-W1-91-2017.

[16] C. Scheiblauer and M. Wimmer, "Out-of-Core Selection and Editing of Huge Point Clouds," *Computers and Graphics*, vol. 35, no. 2, pp. 342–351, Apr. 2011, doi : https://doi.org/10.1016/j.cag.2011.01.004. [Online]. Available: https://www.cg.tuwien.ac.at/research/publications/2011/scheiblauer-2011-cag/

[17] U. Verma and H. Butler, "Plasio," https://plas.io/, 2017, accessed on June 07, 2021.

[18] "Cesiumjs," https://cesium.com/platform/cesiumjs/, 2021, accessed on June 07, 2021.

[19] B. H. Drost and S. Ilic, “Almost constant-time 3D nearest-neighbor lookup using implicit octrees,” *Machine Vision and Applications*, vol. 29, no. 2, pp. 299–311, 2018, doi : https://doi.org/10.1007/s00138-017-0889-4.

[20] M. Pavlovic, K.-N. Bastian, H. Gildhoff, and A. Ailamaki, “Dictionary compression in point cloud data management,” in *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2017, pp. 1–10, doi : https://doi.org/10.1145/3299770.

[21] T. Hackel, J. D. Wegner, and K. Schindler, “Joint Classification and Contour Extraction of Large 3D Point Clouds,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 130, pp. 231–245, 2017, doi : https://doi.org/10.1016/j.isprsjprs.2017.05.012.

[22] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, “RandLA-Net: Efficient semantic segmentation of large-scale point clouds,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 108–11 117, doi : https://doi.org/10.1109/CVPR42600.2020.01112.

[23] K. Liu and J. Boehm, “Classification of Big Point Cloud Data Using Cloud Computing,” *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 40, pp. 553–557, 2015, doi : https://doi.org/10.5194/isprsarchives-XL-3-W3-553-2015.

[24] N. Chehata, L. Guo, and C. Mallet, “Airborne lidar feature selection for urban classification using random forests,” *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 38, no. Part 3, p. W8, 2009.

[25] M. Weinmann, B. Jutzi, and C. Mallet, “Feature relevance assessment for the semantic interpretation of 3d point cloud data,” *ISPRS Annals of the Photogramme-*

try, Remote Sensing and Spatial Information Sciences, vol. 5, p. W2, 2013, doi : https://doi.org/10.5194/isprsannals-II-5-W2-313-2013.

[26] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface Reconstruction from Unorganized Points," *Siggraph Comp. Graph.*, vol. 26, no. 2, pp. 71–78, 1992, doi : https://doi.org/10.1145/142920.134011.

[27] J. Sreevalsan-Nair and B. Kumari, *Local Geometric Descriptors for Multi-Scale Probabilistic Point Classification of Airborne LiDAR Point Clouds*. Springer Cham, Mathematics and Visualization, 2017, pp. 175–200, doi : https://doi.org/10.1007/978-3-319-61358-1_8.

[28] M. Weinmann, B. Jutzi, and C. Mallet, "Semantic 3D scene interpretation: A framework combining optimal neighborhood size selection with relevant features," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 2, no. 3, p. 181, 2014, doi : https://doi.org/10.5194/isprsannals-II-3-181-2014.

[29] J. Sreevalsan-Nair and P. Mohapatra, "Influence of Aleatoric Uncertainty on Semantic Classification of Airborne LiDAR Point Clouds: A Case Study with Random Forest Classifier Using Multiscale Features," in *Proceedings of the IEEE International India GeoScience and Remote Sensing Symposium 2020*, 2020, doi : https://doi.org/10.1109/igarss39084.2020.9323409.

[30] T. Hackel, J. D. Wegner, and K. Schindler, "Fast semantic segmentation of 3D point clouds with strongly varying density," *ISPRS annals of the photogrammetry, remote sensing and spatial information sciences*, vol. 3, no. 3, pp. 177–184, 2016, doi : https://doi.org/10.5194/isprs-annals-iii-3-177-2016.

[31] R. Blomley and M. Weinmann, "Using Multi-scale Features for the 3D Semantic Labeling of Airborne Laser Scanning Data," *ISPRS Annals of Photogram-*

metry, Remote Sensing & Spatial Information Sciences, vol. 4, 2017, doi : https://doi.org/10.5194/isprs-annals-IV-2-W4-43-2017.

[32] F. Rottensteiner, G. Sohn, J. Jung, M. Gerke, C. Baillard, S. Benitez, and U. Breitkopf, "The ISPRS benchmark on urban object classification and 3D building reconstruction," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences I-3*, pp. 293–298, 2012, doi : https://doi.org/10.5194/isprsannals-I-3-293-2012.

[33] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "Kpconv: Flexible and deformable convolution for point clouds," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 6411–6420, doi : https://doi.org/10.1109/iccv.2019.00651.