Enhancement of Scalability in Point Cloud Filtration Algorithms using MapReduce Framework

Muhammad Abdullah[1], Aman Uullah Yasin[2]
CASE, Islamabad
Pakistan
amanyasin@gmail.com
amanyasin@case.edu.pk

ABSTRACT: *In this paper, we have proposed some parallel filtration techniques of point clouds. These techniques are logically based on existing ones present in the open-source Point Cloud Library (PCL). We have used the MapReduce framework provided by Big Data platforms such as Apache Hadoop to address the problem of scalability as well as the completion of processing intensive tasks with relatively cheaper hardware.*

Keywords: MapReduce, Cloud Library, Big Data Platform, Apache Hadoop, Parallel Filteration Techniques

Received: 14 April 2016, Revised 29 May 2016, Accepted 5 June 2016

1. Introduction

In big data, scale up processing on Point Cloud Files (PCF) has many drawbacks. Files can go up to Gigabytes (GB) in size that can take hours to scan on a single machine and sufficient memory is required to run algorithms on these files on a single machine. A single machine with respectable specifications but is dedicated only for a single job. So, the system is reserved during the processing and cannot be used for other purposes and the job at hand still takes a lot of time.

Despite this era being termed as the era of technological advancements, the difference between the rate of growth of computational power of today's computers and rate of growth of file size is alarming. After almost every 3 years, the CPU's computing power increases by a factor of two [1]. The file size compared to this is growing at an amazingly fast pace. It was nearly 20 years ago, that the only common file format was the text file ranging in sizes of KBs (1000 bytes) and now file sizes have gone up to giga (10^9 bytes), tera (10^{12} bytes) and peta (10^{15} bytes).

File size is increasing rapidly but the algorithms are not optimizing well to the file sizes in terms of processing and memory usage efficiency. The approach of using faster CPUs but with traditional algorithms is has shown improvement at a very small scale. Moreover, newer format always require more complicated decoding algorithms and lead to even longer processing times. Thus

the only feasible way that remains in order to attain large speed up with the current computational power is to split the job at hand onto several machines and execute it in parallel.

2. Background

The Point Cloud Library (PCL) and Hadoop platform has been used for our research. The Point Cloud Library (PCL) is a large scale cross-platform open source C++ programming library which implements a large number of point cloud universal algorithms and efficient data structures. The PCL has a variety of 3D point cloud data processing algorithm set. The Point Cloud data is taken by the PCL library in the PCD (Point Cloud Data) format. PCD file formats might have different revision numbers, prior to the release of Point Cloud Library (PCL) version 1.0.

On the other hand Hadoop is a framework (consisting of software libraries) which simplifies the processing of data sets distributed across clusters of servers. Two of the main components of Hadoop are HDFS and MapReduce. HDFS is the file system that is used by Hadoop to store all the data on where as MapReduce is the framework that orchestrates all of Hadoop's activities. It handles the assignment of work to different nodes in the cluster.

3. Methodology

We have studied three PCL algorithms which are Pass-Through Filter, Conditional Outlier Removal Filter and Voxel Grid Filter. We have shortlisted these algorithms for the implementation on the MapReduce paradigm. The sequential versions of the algorithms implemented by PCL have been discussed in detail in each sub-chapter after which our proposed approach for the respective algorithm has been put forward to run on the MapReduce paradigm. Section A provides insight into the working of the Pass-Through Filter on PCL along with the parallel solution designed by us to execute on the MapReduce framework. Similarly Section B and C discuss the Conditional Outlier Removal and Voxel Grid Filters respectively.

3.1 Pass-Through Filter

The Pass-Through Filter algorithm implemented in PCL is designed for the filtering of points inside or outside a given range (limits) specified by the user along a certain dimension (field) that is x-axis, y-axis or z-axis as specified by the user. This filter iterates over the entire point cloud once and filters out all non-infinite points lying outside the range specified by the user.

The implementation of the Pass-Through Filter on the MapReduce paradigm has been divided into three modules namely the mapper, combiner and reducer procedures. The first phase namely the Pass-Through Mapper will iterate over designated chunk of the point cloud and check whether each point is inside or outside the range across the field both specified by the user. During this process, the mapper will emit tab-delimited key-value pairs where the key portion will be a combination of the x-coordinate, y-coordinate and z-coordinate values of the point sorted by the Partitioner function [24].The corresponding value portion will consist of a status flag which will hold a positive value "true" if the point lies within the specified range or a negative value "false" if the point lies outside the specified range. The key has been chosen as a combination of the three coordinate values as key should be unique for every record [19] and it is understood that each point will occur only once on that same position in the point cloud dataset. Once the whole dataset has been iterated, the combiner phase of the algorithm will come into action.

Algorithm 1(b)(i) Pseudo code for Pass-Through Mapper

procedure Pass-Through Mapper

1.Set Filter Field Name

2.Set Filter Field Limits

3.Set status = **false**

4.for each point *I* in point cloud chunk:

5. if *I* lies within range:

6.status = **true**

7.**else**

8.Status = **false**

9.emit(key, value) ·

end procedure

The second phase namely the Pass-Through Combiner will iterate over all the key-value pairs emitted by the Pass-Through Mapper and check the status flag of all the points during the process. If the status flag for a particular point is positive "true", the combiner will emit that particular key-value pair in the same format to the reducer phase otherwise the point will simply be ignored. The combiner's output format should match its input format for it to properly work in the MapReduce paradigm [19]. The combiner's main purpose is to perform local filtration of points so as to minimize the records going to the reducer phase which will be running on a single machine thereby increasing performance by avoiding bottlenecks in transmission over the network.

Algorithm 1(b)(ii) Pseudo code for Pass-Through Combiner

procedure Pass-Through Combiner

1. **for** each (key, value):

2. **if** status is **true**:

3. emit(key2, value2)

end procedure

The third and final phase namely the Pass-Through Reducer phase will iterate over all the key-value pairs emitted by the combiner and will convert the input into the same format which was read by the mapper from the point cloud file by ignoring the status flag and writing down all the coordinate values i.e. x-coordinate, y-coordinate and z-coordinate values separated by spaces which will be consequently written to the resulting point cloud file.

Algorithm 1(b)(iii) Pseudo code for Pass-Through Reducer

procedure Pass-Through Reducer

1. **for** each (key, value):

2. emit(x-value, y-value, z-value)·

end procedure

3.2 Conditional Outlier Removal Filter

The Conditional Outlier Removal Filter implemented in PCL works in exactly the same way as the Pass-Through Filter except the fact that multiple conditions can be stated by the user for the purpose of filtration. In simple words, the algorithm can crop points outside ranges specified across multiple dimensions at the same time or apply multiple Pass-Through Filters. Furthermore this algorithm unlike the Pass-Through Filter is not only intended for filtration based upon spatial values but can also filter points based upon their color 'RGB' and curvature values [2].

The implementation of the Conditional Outlier Removal Filter on the MapReduce paradigm has been divided into three modules namely the mapper, combiner and reducer procedures. The first phase namely the Conditional Outlier Removal Mapper will iterate over designated chunk of the point cloud and check whether each point is inside or outside the range across the field both specified by the user. The operation mentioned above will be repeated for all the conditions specified by the user. During this process, the mapper will emit tab-delimited key-value pairs where the key portion will be a combination of the x-coordinate, y-coordinate and z-coordinate values of the point sorted by the Partitioner function [24]. The corresponding value portion will consist of a status flag which will hold a positive value "true" if the point lies within the specified range or a negative value "false" if the point lies outside the specified range. The key has been chosen as a combination of the three coordinate values as key should be unique for every record [19] and it is understood that each point will occur only once on that same position in the point cloud dataset. The PCL version of this algorithm also supports filtration based upon the color 'RGB' and curvature values [2] but our MapReduce version of the algorithm will only support filtration for spatial values as it will deal with point clouds in the XYZ format (.xyz) which will be discussed in Chapter 5. Once the whole dataset has been iterated, the combiner phase of the algorithm will come into action.

The second phase namely the Conditional Outlier Removal Combiner will iterate over all the key-value pairs emitted by the Conditional Outlier Removal Mapper and check the status flag of all the points during the process. If the status flag for a

Algorithm 2(b)(i) Pseudo code for Conditional Outlier Removal Mapper

procedure Conditional Outlier Removal Mapper

1. **for** total number of conditions (n):

2. Set Filter Field Name

3. Set Filter Field Limits

4. Set status = **false**

5. **for** each point I in point cloud chunk:

6. **for** total number of conditions (n):

7. **if** *I* lies within range of n:

8. status is **true**

9. **else**

10. status is **false**

11. emit(key, value) ·

end procedure

particular point is positive "true", the combiner will emit that particular key-value pair in the same format to the reducer phase otherwise the point will simply be ignored. The combiner's output format should match its input format for it to properly work in the MapReduce paradigm [19].

Algorithm 2(b)(ii) Pseudo code for Conditional Outlier Removal Combiner

procedure Conditional Outlier Removal Combiner

1. **for** each (key, value):

2. **if** status is **true:**

3. emit(key2, value2)

end procedure

The third and final phase namely the Conditional Outlier Removal Reducer phase will iterate over all the key-value pairs emitted by the combiner and will convert the input into the same format which was read by the mapper from the point cloud file by ignoring the status flag and writing down all the coordinate values i.e. x-coordinate, y-coordinate and z-coordinate values separated by spaces which will be consequently written to the resulting point cloud file.

Algorithm 1(b)(iii) Pseudo code for Conditional Outlier Removal Reducer

procedure Conditional Outlier Removal Reducer

1. **for** each (key, value):

2. emit(x-value, y-value, z-value)

end procedure

3.3 Voxel Grid Filter

The Voxel Grid Filter implemented in PCL is also known by the name of the Down Sampling Filter. There are two categories for this filter in PCL namely the Voxel Grid and Approximate Voxel Grid filters but our main concern will only be with the simple Voxel Grid Filter. This filter takes in the input point cloud in the form of voxels where voxels are small 3D cubes whose dimensions are specified by the user. The filter will then compute the spatial centroid for all the individual voxels representing the point cloud and will replace all the points residing in that particular voxel by that one centroid henceforth down sampling or filtering the

entire pointcloud [2], [19].

The implementation of the Voxel Grid Filter on the MapReduce paradigm has been divided into three modules namely the mapper, combiner and reducer procedures. The first phase namely the Voxel Grid Mapper will iterate over designated chunk of the point cloud and will classify each and every point to a voxel by assigning them a voxel identification number relative to the point (0, 0, 0). In this case the origin will be acting as the global point of reference and the numbering of each voxel ID will be based purely on this point. For negative values, the voxel IDs will also be negative. During this process, the mapper will emit tab-delimited key-value pairs where the key portion will be a combination of the x-value, y-value and z-value IDs of the voxel to which the corresponding point belongs which will be in turn sorted by the Partitioner function [24]. The corresponding value portion will consist of the coordinates of the point itself i.e. its x-coordinate, y-coordinate and z-coordinate values and the occurrence of that point which will be fixed for all points at 1. Once the whole dataset has been iterated, the combiner phase of the algorithm will come into action.

Algorithm 3(b)(i) Pseudo code for Voxel Grid Mapper

procedure Voxel Grid Mapper

1. Set Voxel Dimensions
2. **for** each point I in point cloud chunk:
3. compute x dimension ID relative to origin
4. compute y dimension ID relative to origin
5. compute z dimension ID relative to origin
6. emit(key, value)

end procedure

The second phase namely the Voxel Grid Combiner will iterate over all the key-value pairs emitted by the Voxel Grid Mapper and will sum up the x-coordinate, y-coordinate and z-coordinate values separately for all the points belonging to the same voxel during the process. In addition to this, the total number of points in each voxel will also be computed by adding up the occurrence values of all the points in that particular voxel which will always be equal to 1. The combiner will then consequently emit one key-value pair for each voxel. The combiner's output format should match its input format for it to properly work in the MapReduce paradigm [19].

Algorithm 3(b)(ii) Pseudo code for Voxel Grid Combiner

procedure Voxel Grid Combiner

1. **for** each (key, value):
2. **if** voxel ID is unchanged:
3. Add x-value to net of x coordinates
4. Add y-value to net of y coordinates
5. Add z-value to net of z coordinates
6. Increment counter for points in voxel
7. **else:**
8. Reset net of x coordinates to zero
9. Reset net of y coordinates to zero
10. Reset net of z coordinates to zero
11. Reset counter to one
12. emit(key2, value2)

end procedure

The third and final phase namely the Voxel Grid Reducer phase will iterate over all the key-value pairs emitted by the combiner and will compute the centroid for each voxel by dividing the total x-coordinate, y-coordinate and z-coordinate values with the total number of points in the particular voxel both of which have been calculated and emitted in key-value pair by the combiner. It will then convert the input into the same format which was read by the mapper from the point cloud file by ignoring the key portion containing the voxel ID and writing down all the coordinate values i.e. x-coordinate, y-coordinate and z-coordinate values separated by spaces which will be consequently written to the resulting point cloud file.

Algorithm 3(b)(iii) Pseudo code for Voxel Grid Reducer

procedure Voxel Grid Reducer

1. **for** each (key, value):

2. **if** voxel ID is unchanged:

3. compute x coordinate of centroid

4. compute y coordinate of centroid

5. compute z coordinate of centroid

6. **else:**

7. emit(x-centroid, y-centroid, z-centroid)

end procedure

4. Experimental Result

This section covers the whole evaluation portion of our Research. Section A describes all the testing constraints such as the file format, size and filtration conditions that must be kept constant within the testing systems to ensure verifiable and valid results. In addition to this it also gives a description of the nature of the input datasets chosen for experimentation. Section B describes the environmental setup of the working environment. The first sub-section describes the PCL environment on a single machine. The last sub-section defines the setup of Apache Hadoop on a cluster of four machines. Section C gives the statistical results of the experiments conducted. It illustrates each algorithm's result separately in sub-sections and provides the results of both systems one after the other in sub-chapters for the purpose of comparative analysis.

4.1 Testing Constraints and Input Dataset

During the conduction of the experiments, it was ensured that the following attributes must remain the same for both the testing environments that is PCL and the Hadoop multi-node cluster:

- The size and nature of the dataset.
- The filtration thresholds for each algorithm.
- The file format must remain the same for each individual system.

The datasets were taken from online repositories of universities as well as organizations doing projects on point clouds [25], [26]. Point Clouds can be represented in various file formats. The two formats compatible with our systems are the XYZ format and the PCD (Point Cloud Data) format. The PCD format is intended for the PCL as it accepts point clouds in this format only. PCD file formats might have different revision numbers, prior to the release of Point Cloud Library (PCL) version 1.0. These are numbered with PCD_Vx (e.g., PCD_V5, PCD_V6, PCD_V7, etc) and represent version numbers 0.x for the PCD file. The official entry point for the PCD file format in PCL however should be version *0.7* (*PCD_V7*). The PCD file header contains headers such as *Version, Fields, Size, Type, Count, Width, Height, Viewpoint, Points and Data* respectively. The data is stored in two formats that are either in ASCII or in binary format. The XYZ format is a more simple representation of a point cloud. It is a simple text file comprising of three columns for each of the x, y and z dimensions. Each row in this file signifies a record or point entry in that point cloud and the x, y and z coordinates are provided in floating-point numbers. Figure 5.1 illustrates the layout of a typical XYZ file.

Many problems were faced during the collection of the datasets. First of all the XYZ format point clouds were not readily

available in large numbers from a single repository. Secondly large point clouds had been uploaded in chunks with each chunk containing additional headers. Each point cloud would have up to 40 chunks on average. In order to view the point cloud in 3D, we had to feed it into an online viewer by the name of LiDAR viewer [28]. This viewer only accepted files with the following conditions:

- No special characters or comments.
- Columns should be space-delimited.

```
-21.894799 33.314639 30.003743
-22.504399 33.345119 30.003743
-23.037799 33.340039 30.003743
-23.601679 33.223199 30.003743
-24.104599 33.182559 30.003743
-24.561799 33.248599 30.003743
-24.998679 33.075879 30.003743
-25.511759 33.065719 30.003743
-25.933399 32.887919 30.003743
-26.537919 32.715199 30.003743
-26.863039 32.359599 30.003743
-27.091639 31.897319 30.003743
-27.279599 31.409639 30.003743
-27.320239 30.891479 30.003743
-27.330399 30.434279 30.003743
-27.162759 29.966919 30.003743
-27.304999 29.474159 30.003743
-27.320239 28.940759 30.003743
-27.254199 28.392119 30.003743
-27.249119 27.863799 30.003743
```

Figure 1. This figure provides a snapshot of a sample XYZ file containing point cloud data

Point Cloud Dataset Information Table				
Sr. #	**File Name**	**Data Points**	**File Size**	**File Format**
1	Pcmhouse1	123,748	4.0 MB	XYZ
2		(0.12 million)	0.886 MB	PCD
3	Pcmhouse4	480,852	15.4 MB	XYZ
4		(0.49 million)	3.4 MB	PCD
5	Oakland	1,514,625	30.9 MB	XYZ
6		(1.51 million)	9.4 MB	PCD
7	Building	4,572,428	211.6 MB	XYZ
8		(4.57 million)	27.5 MB	PCD

Table 1. The table gives a list of all the datasets along with their details that have been selected for testing

Due to all the above mentioned reasons, we were able to locate four XYZ datasets fit for our experimentation. The datasets were chosen in such a way that each dataset chosen would consist of thrice as many points as the previously chosen dataset so as to get visually clear graphical readings later on in the experiment. This in turn would make comparisons based on scalability a lot simpler. In order to validate the testing constraints for out experiment, we had to feed the same datasets to the PCL as well but the problem was that PCL only accepts PCD format. To resolve this issue we used PCL's basic XYZ to PCD conversion tool

available on GitHub [27] to convert our XYZ datasets to PCD format. The huge difference between the sizes of the XYZ and PCD formats for the same dataset is due to the PCD format taking point data in compressed binary. XYZ format on the other hand takes the point data in ASCII format. The shortlisted datasets along with their details have been provided below:

Once all the datasets had been shortlisted, the next task was to set the filtration thresholds for the individual datasets as well as the algorithms. A total of three tests were conducted for each algorithm on each dataset making it a total of 12 tests for each algorithm and a net total of 36 tests for each of the systems that is PCL and Hadoop. A grand total of 72 tests were conducted for the overall project. The criteria for the conducted tests have been provided in the figure below:

Experimental Testing Criteria		
Test #	**File Name**	**Filter Conditions**
	Pass-Through Filter	
1-3	Pcmhouse1	Filter Field = Z – Dimension Filter Limits: (-50.0 <= Z <= 50.0)
4-6	Pcmhouse4	Filter Field = Z – Dimension Filter Limits: (-50.0 <= Z <= 50.0)
7-9	Oakland	Filter Field = X – Dimension Filter Limits: (-150.0 <= X<= -50.0)
10-12	Building	Filter Field = X – Dimension Filter Limits: (-315250.0 <= X<= 315300.0)
	Conditional Outlier Removal Filter	
1-3	Pcmhouse1	**Condition 1** Filter Field = X – Dimension Filter Limits: (0.0 <= X<= 50.0) **Condition 2** Filter Field = Y – Dimension Filter Limits: (0.0 <= Y<= 50.0) **Condition 3** Filter Field = Z – Dimension Filter Limits: (-50.0 <= Z <= 90.0)
4-6	Pcmhouse4	**Condition 1** Filter Field = X – Dimension Filter Limits: (0.0 <= X<= 50.0) **Condition 2** Filter Field = Y – Dimension Filter Limits: (0.0 <= Y<= 50.0) **Condition 3** Filter Field = Z – Dimension Filter Limits: (-50.0 <= Z <= 90.0)
7-9	Oakland	**Condition 1** Filter Field = X – Dimension Filter Limits: (-150.0 <= X<= -50.0) **Condition 2** Filter Field = Y – Dimension Filter Limits: (-250.0 <= Y<= -150.0) **Condition 3** Filter Field = Z – Dimension Filter Limits: (-10.0 <= Z <= 0.0)
10-12	Building	**Condition 1** Filter Field = X – Dimension Filter Limits: (315250.0 <= X <= 315300.0) **Condition 2** Filter Field = Y – Dimension Filter Limits: (4834350.0 <= Y <= 4834450.0) **Condition 3** Filter Field = Z – Dimension Filter Limits: (85.0 <= Z <= 95.0)
	Voxel Grid Filter	
1-3	Pcmhouse1	Voxel Size: X = 5.0, Y = 5.0, Z = 5.0 (Cube of 5 x 5 x 5)
4-6	Pcmhouse4	Voxel Size: X = 5.0, Y = 5.0, Z = 5.0 (Cube of 5 x 5 x 5)
7-9	Oakland	Voxel Size: X = 5.0, Y = 5.0, Z = 5.0 (Cube of 5 x 5 x 5)
10-12	Building	Voxel Size: X = 5.0, Y = 5.0, Z = 5.0 (Cube of 5 x 5 x 5)

Table 2. The table gives a list of all the tests conducted along with the testing conditions. Testing conditions for both PCL and Apache Hadoop are the same.

4.2 Hardware & Software Environment

1) Point Cloud Library (PCL)

The PCL environment was setup on a single machine that is the Dell Inspiron N5050 laptop as mentioned in Table 1.1. The system had the Linux Ubuntu 14.04 (64-bit) operating system running on it. After the download and installation of the PCL binaries and source libraries from [2], individual projects were implemented for the algorithms as described in Chapter 4. CMake v3.2.1 was installed in order to compile the PCL projects into executable. After compilation, the projects were run using the built-in GNU C++compiler in Ubuntu. After documenting the results of the three tests for each algorithm, the average processing time of the three tests was computed for each dataset. A graph was then plotted of the average processing time of the algorithm against the number of data points in the dataset using an online Point Plotter Tool [29].

2) Hadoop Multi-Node Cluster

The Apache Hadoop multi-node cluster environment was setup using 4 Dell Desktop PCs as described in Table 1.1. The computers were connected to each other via LAN (Local Area Network) through the TP-Link switch as described in Table 1.1. One of the PCs was the designated the role of master while the rest of the three were acting as the slaves. The input datasets and the necessary source code files (Python scripts) were uploaded to the HDFS via the master node. These scripts were in turn converted to Mapper, Combiner and Reducer scripts by the Hadoop Streamer commands [24]. Once the tests were conducted, the output point clouds were imported from the HDFS back to the local disk through the Hadoop commands. After documenting the results of the three tests for each algorithm, the average processing time of the three tests was computed for each dataset. A graph was then plotted of the average processing time of the algorithm against the number of data points in the dataset using an online Point Plotter Tool [29].

4.3 Statistical Analysis of Conducted Experiments

1) Pass-Through Filter

A total of 12 tests were conducted for PCL and another 12 for Hadoop. In order to find the scalability factor the relation between the average execution time of the tests three and the number of data points in the point cloud dataset was recorded. The following sub-sections provide the average execution times corresponding to each point cloud dataset along with their graphical representations. Moreover, a graph was plotted on Microsoft Excel 2007 of the gaining average processing times for each test relative to the time of the test conducted on the smallest dataset for both PCL and Hadoop against the number of data points in the dataset. This graph has been provided in the last sub-section and is intended to depict the comparison of scalability of PCL and Hadoop. The formula for finding the gain in execution time is as follows:

a) Point Cloud Library Results (Single Machine)

Pass-Through Filter (PCL) Data			
Test #	**File Name**	**Data Points**	**Average Execution Time (milliseconds)**
1-3	Pcmhouse1	123,748 (0.12 million)	103.06
4-6	Pcmhouse4	480,852 (0.49 million)	365.76
7-9	Oakland	1,514,625 (1.51 million)	1006.56
10-12	Building	4,572,428 (4.57 million)	4057.36

Table 3. The table provides the average execution times of the Pass-Through filtration algorithm on each of the point cloud datasets on PCL

$$Gain = [(Xcurrent - Xprevious) / Xprevious] + 1 \quad (1)$$

Where Xcurrent = Current Execution Time
X previous = Previous Execution Time
If (X Previous = X Current) then Gain = 1

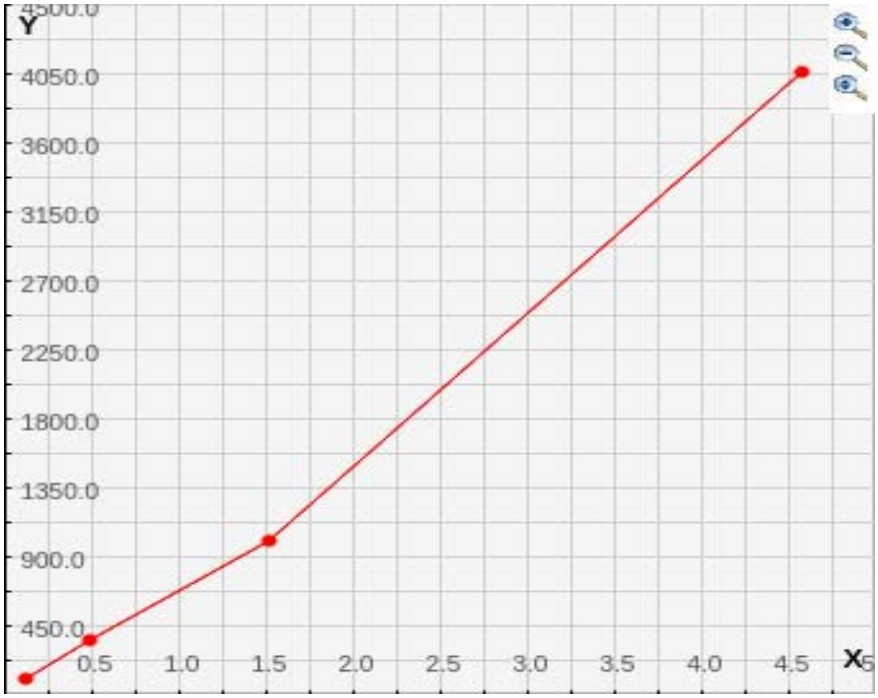

Figure 2. The figure illustrates the relationship between the average execution time in milliseconds (y-axis) and the number of data points divided by 10^6 (x-axis) for the Pass-Through Filter on PCL

b) Hadoop MapReduce Results (Multi-Node Cluster)

Pass-Through Filter (Hadoop) Data			
Test #	**File Name**	**Data Points**	**Average Execution Time (seconds)**
1-3	Pcmhouse1	123,748 (0.12 million)	14
4-6	Pcmhouse4	480,852 (0.49 million)	15.3333
7-9	Oakland	1,514,625 (1.51 million)	18
10-12	Building	4,572,428 (4.57 million)	22

Table 4. The table provides the average execution times of the Pass-Through filtration algorithm on each of the point cloud datasets on Hadoop

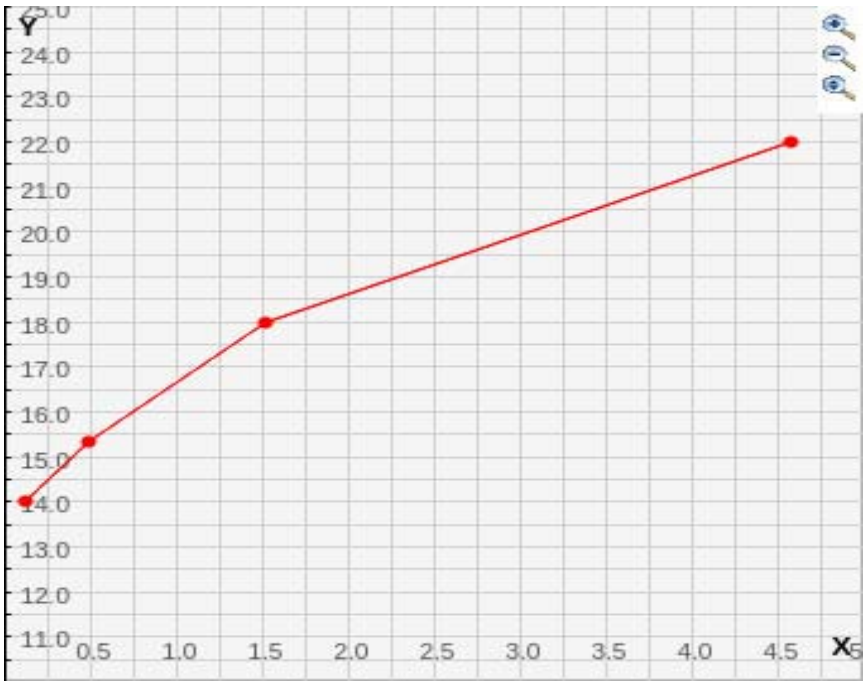

Figure 3. The figure illustrates the relationship between the average execution time in seconds (y-axis) and the number of data points divided by 10^6 (x-axis) for the Pass-Through Filter on Hadoop

c) Scalability Comparison of PCL & Hadoop Normalized Execution Times

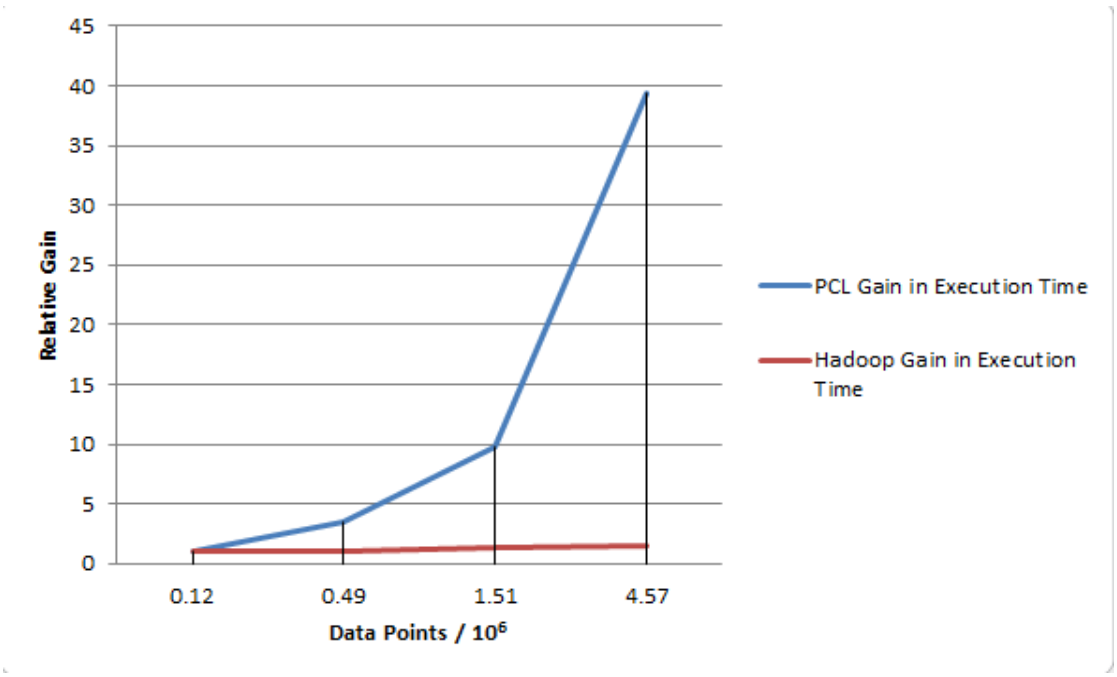

Figure 4. The figure illustrates the relationship between the gain in execution times (y-axis) relative to the execution time taken on the smallest dataset and the number of data points divided by 10^6 (x-axis) for the Pass-Through Filter on Hadoop and PCL

Pass-Through Filter (Scalability) Data				
Test #	**File Name**	**Data Points**	**Gain (PCL)**	**Gain (Hadoop)**
1-3	Pcmhouse1	123,748 (0.12 million)	1	1
4-6	Pcmhouse4	480,852 (0.49 million)	3.549000582	1.095235714
7-9	Oakland	1,514,625 (1.51 million)	9.766737823	1.285714286
10-12	Building	4,572,428 (4.57 million)	39.36891131	1.571428571

Table 5. The table provides the gain in average execution times relative to the execution time taken on the smallest dataset of the Pass-Through filtration algorithm on each of the point cloud datasets on Hadoop and PCL.

2) Conditional Outlier Removal Filter

A total of 12 tests were conducted for PCL and another 12 for Hadoop. In order to find the scalability factor the relation between the average execution time of the tests three and the number of data points in the point cloud dataset was recorded. The following sub-sections provide the average execution times corresponding to each point cloud dataset along with their graphical representations. Moreover, a graph was plotted on Microsoft Excel 2007 of the gain in average processing times for each test relative to the time of the test conducted on the smallest dataset for both PCL and Hadoop against the number of data points in the dataset. This graph has been provided in the last sub-section and is intended to depict the comparison of scalability of PCL and Hadoop. The formula for finding the gain in execution time is as follows:

a) Point Cloud Library Results (Single Machine)

$$Gain = [(Xcurrent - Xprevious) / Xprevious] + 1 \quad (1)$$

where Xcurrent = Current Execution Time
X previous = Previous Execution Time
If (X previous = X current) then Gain = 1

Conditional Outlier Removal Filter (PCL) Data			
Test #	**File Name**	**Data Points**	**Average Execution Time (milliseconds)**
1-3	Pcmhouse1	123,748 (0.12 million)	102.20
4-6	Pcmhouse4	480,852 (0.49 million)	363.45
7-9	Oakland	1,514,625 (1.51 million)	492.84
10-12	Building	4,572,428 (4.57 million)	2215.76

Table 6. The table provides the average execution times of the Conditional Outlier Removal filtration algorithm on each of the point cloud datasets on PCL

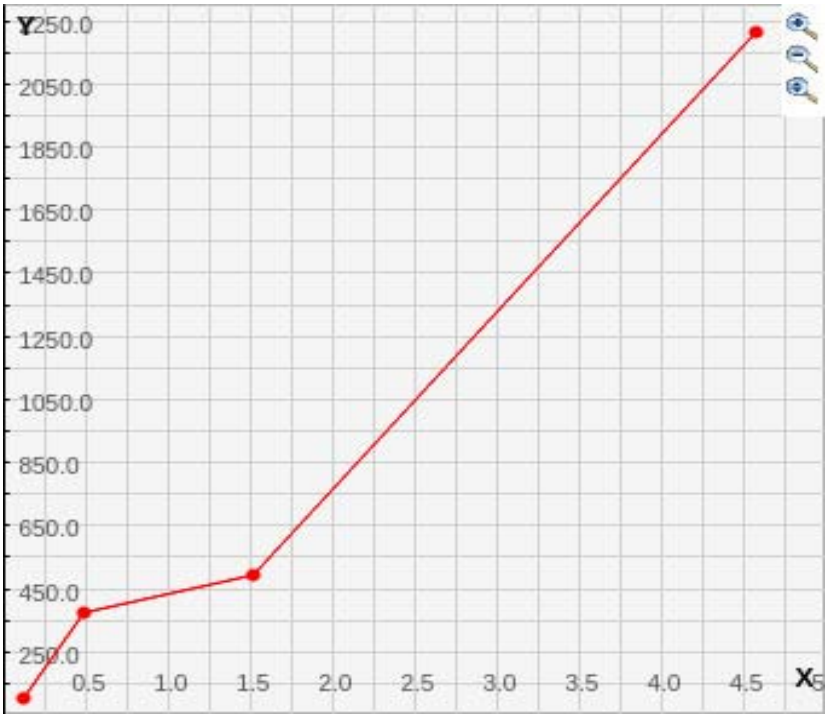

Figure 5. The figure illustrates the relationship between the average execution time in milliseconds (y-axis) and the number of data points divided by 10^6 (x-axis) for the Conditional Outlier Removal Filter on PCL

b) Hadoop MapReduce Results (Multi-Node Cluster)

Conditional Outlier Removal Filter (Hadoop) Data			
Test #	**File Name**	**Data Points**	**Average Execution Time (seconds)**
1-3	Pcmhouse1	123,748 (0.12 million)	14
4-6	Pcmhouse4	480,852 (0.49 million)	15
7-9	Oakland	1,514,625 (1.51 million)	16.3333
10-12	Building	4,572,428 (4.57 million)	25.3333

Table 7. The table provides the average execution times of the Conditional Outlier Removal filtration algorithm on each of the point cloud datasets on Hadoop

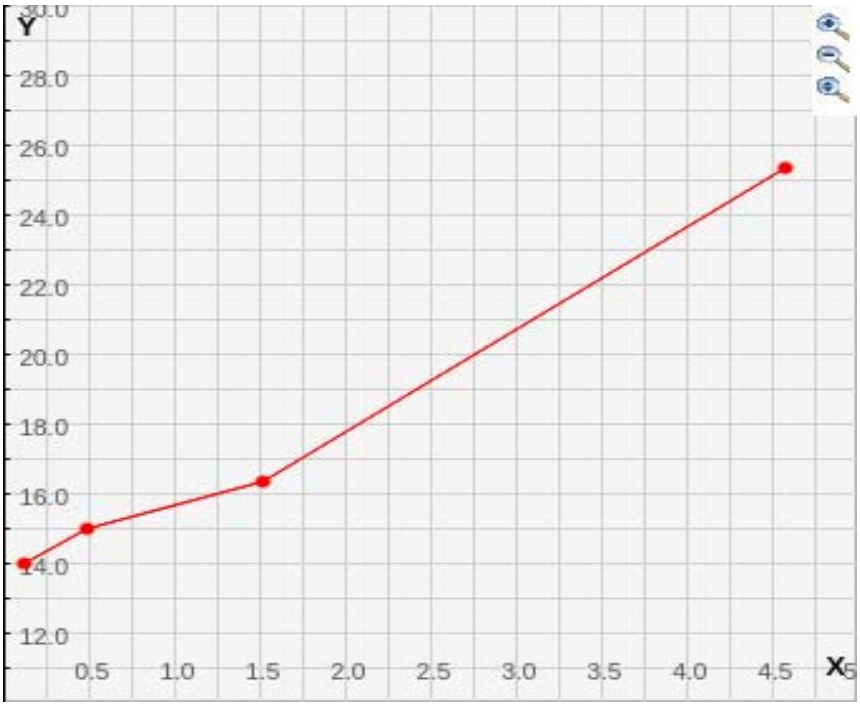

Figure 6. The figure illustrates the relationship between the average execution time in seconds (y-axis) and the number of data points divided by 10^6 (x-axis) for the Conditional Outlier Removal Filter on Hadoop

c) Scalability Comparison of PCL & Hadoop Normalized Execution Times

Conditional Outlier Removal Filter (Scalability) Data				
Test #	**File Name**	**Data Points**	**Gain (PCL)**	**Gain (Hadoop)**
1-3	Pcmhouse1	123,748 (0.12 million)	1	1
4-6	Pcmhouse4	480,852 (0.49 million)	3.556262231	1.071428571
7-9	Oakland	1,514,625 (1.51 million)	4.822309198	1.166664286
10-12	Building	4,572,428 (4.57 million)	21.68062622	1.809521429

Table 8. The table provides the gain in average execution times relative to the execution time taken on the smallest dataset of the Conditional Outlier Removal filtration algorithm on each of the point cloud datasets on Hadoop and PCL

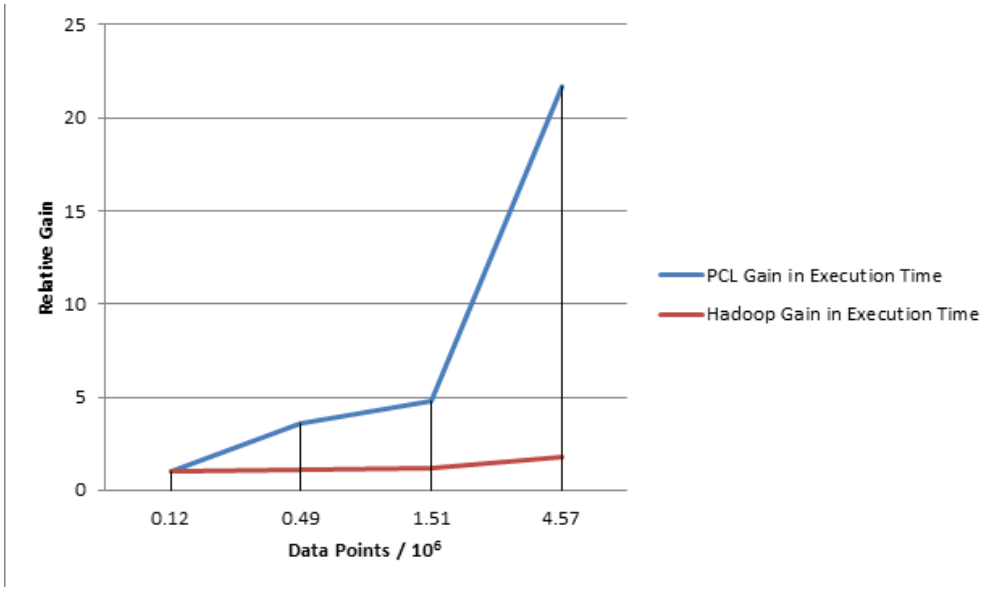

Figure 7. The figure illustrates the relationship between the gain in execution times (y-axis) relative to the execution time taken on the smallest dataset and the number of data points divided by 10^6 (x-axis) for the Conditional Outlier Removal Filter on Hadoop and PCL

3) Voxel Grid Filter

A total of 12 tests were conducted for PCL and another 12 for Hadoop. In order to find the scalability factor the relation between the average execution time of the tests three and the number of data points in the point cloud dataset was recorded. The following sub-sections provide the average execution times corresponding to each point cloud dataset along with their graphical representations. Moreover, a graph was plotted on Microsoft Excel 2007 of the gain in average processing times for each test relative to the time of the test conducted on the smallest dataset for both PCL and Hadoop against the number of data points in the dataset. This graph has been provided in the last sub-section and is intended to depict the comparison of scalability of PCL and Hadoop. The formula for finding the gain in execution time is as follows:

a) Point Cloud Library Results (Single Machine)

Voxel Grid Filter (PCL) Data			
Test #	**File Name**	**Data Points**	**Average Execution Time (milliseconds)**
1-3	Pcmhouse1	123,748 (0.12 million)	49.47
4-6	Pcmhouse4	480,852 (0.49 million)	164.45
7-9	Oakland	1,514,625 (1.51 million)	513.58
10-12	Building	4,572,428 (4.57 million)	1491.08

Table 9. The table provides the average execution times of the Voxel Grid filtration algorithm on each of the point cloud datasets on PCL

$$Gain = [(Xcurrent - Xprevious) / Xprevious] + 1 \quad (1)$$

where Xcurrent = Current Execution Time

X previous = Previous Execution Time

If (X previous = X current) then Gain = 1

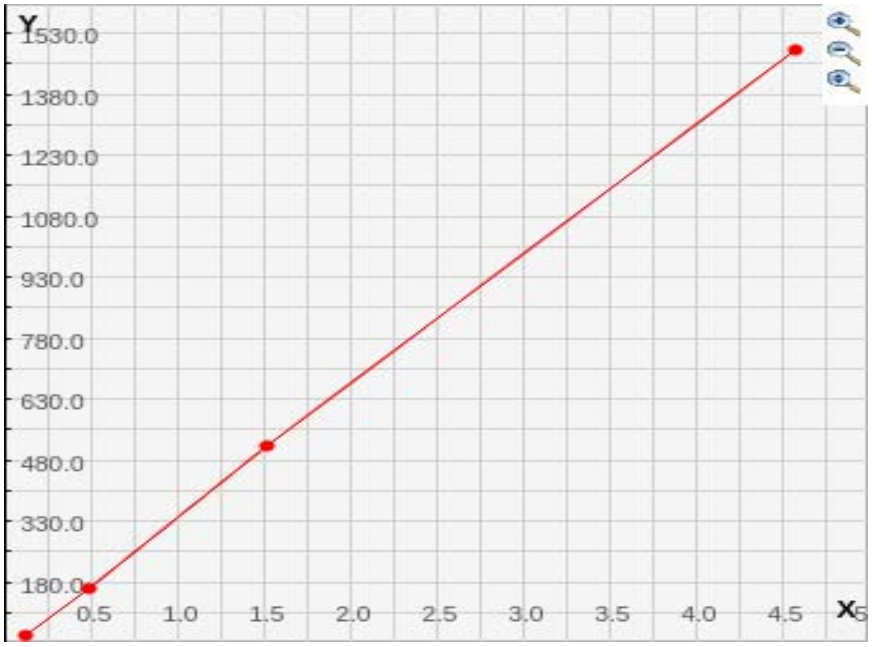

Figure 8. The figure illustrates the relationship between the average execution time in milliseconds (y-axis) and the number of data points divided by 10^6 (x-axis) for the Voxel Grid Filter on PCL

b) Hadoop MapReduce Results (Multi-Node Cluster)

Voxel Grid Filter (Hadoop) Data			
Test #	**File Name**	**Data Points**	**Average Execution Time (seconds)**
1-3	Pcmhouse1	123,748 (0.12 million)	14
4-6	Pcmhouse4	480,852 (0.49 million)	15.3333
7-9	Oakland	1,514,625 (1.51 million)	18
10-12	Building	4,572,428 (4.57 million)	28.3333

Table 10. The table provides the average execution times of the Voxel Grid filtration algorithm on each of the point cloud datasets on Hadoop

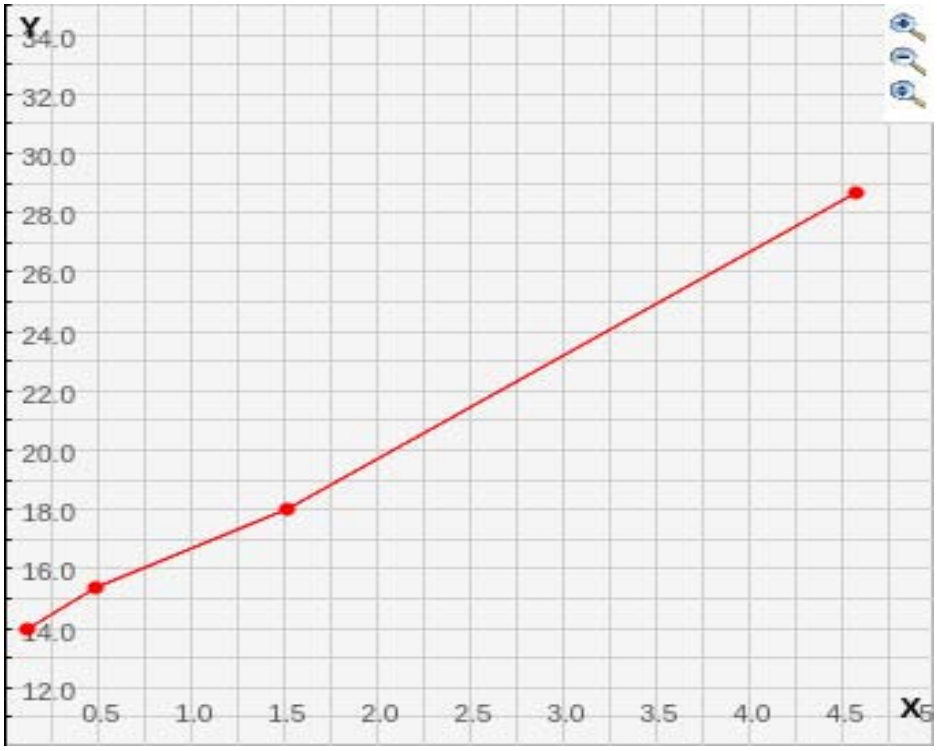

Figure 9. The figure illustrates the relationship between the average execution time in seconds (y-axis) and the number of data points divided by 10^6 (x-axis) for the Voxel Grid Filter on Hadoop

c) Scalability Comparison of PCL & Hadoop Normalized Execution Times

Voxel Grid Filter (Scalability) Data				
Test #	**File Name**	**Data Points**	**Gain (PCL)**	**Gain (Hadoop)**
1-3	Pcmhouse1	123,748 (0.12 million)	1	1
4-6	Pcmhouse4	480,852 (0.49 million)	3.324236911	1.095235714
7-9	Oakland	1,514,625 (1.51 million)	1.285714286	10.38164544
10-12	Building	4,572,428 (4.57 million)	30.14109561	2.023807143

Table 11. The table provides the gain in average execution times relative to the execution time taken on the smallest dataset of the Voxel Grid filtration algorithm on each of the point cloud datasets on Hadoop and PCL

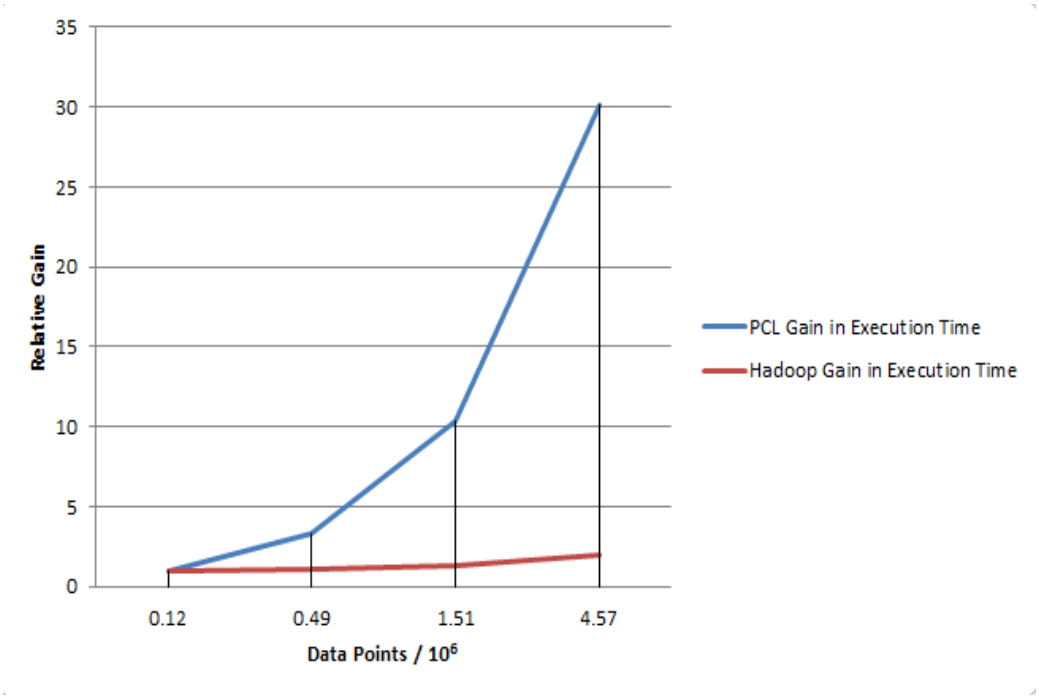

Figure 10. The figure illustrates the relationship between the gain in execution times (y-axis) relative to the execution time taken on the smallest dataset and the number of data points divided by 10^6 (x-axis) for the Voxel Grid Filter on Hadoop and PCL

5. Conclusion

We have presented a parallel solution to the filtration algorithms implemented in PCL. As suggested by our experimental results, the proposed algorithms scale very well as the size of the dataset increases. This phenomenon is not prominent in the results conducted on the PCL. In our experiments, we have measured scalability as the rise of the processing time against the increase in size of the dataset (gradient of the graph) of a single test relative to the previous test. In case of PCL, the processing time increases by a factor of 3 for every test conducted. Hadoop on the other hand has an increase in the processing time by a factor of less than 1.5 for every test. Such a great difference clearly shows the superiority in terms of scalability in the Apache Hadoop platform. The main reason behind this improved scalability factor is due to the effectiveness and efficiency of the MapReduce framework in parallelizing the programming logic into multiple processes and hence distributing the workload across the machines connected in the multi-node cluster supervised by the Apache Hadoop platform. The proposed approach is flexible enough to come across machine failures and network connection problems by re-distributing the load according to the status of the machines in the clusters. Hadoop will generate more fruitful results when tested on a large farm of machines and with the input dataset in the range of 10 to 50 Gigabytes in size. Due to limited resources, this phenomenon was beyond the scope of our project. Although in the single-process performance comparison, PCL wins from Hadoop by a great margin for the following reasons:

- C/C++ supports fast and efficient data structure creation and manipulation as compared to Python.
- The Boost library in PCL further enhances the performance of data storage and memory utilization for PCL.
- The Hadoop Streamer process is an additional significant overhead to the overall execution time as the code written in Python has to be converted to a MapReduce compatible language that is Java.
- Hadoop replication of files across various machines in the cluster adds a 4 times multiplier (based on custom cluster setting) to its already large storage footprint. This in turn creates I/O bottlenecks which
- Account for a major portion of the execution time [30].
- The PCD file format for PCL takes in the data in compressed binary format while our approach on the MapReduce framework

takes the file in ASCII format. This leads to more time being spent in file reading and writing.

References

[1] Moore's Law - Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Moore%27s_law

[2] PCL – Point Cloud Library (PCL), http://pointclouds.org

[3] CMake, http://www.cmake.org

[4] Yiu, Ling ., Zhong, Ruofei (2014). Buildings and Terrain of Urban Area Point Cloud Segmentation based on PCL, *In*: 35th International Symposium on Remote Sensing of Environment (ISRSE35), IOP Publishing 2014.

[5] FLANN (Fast Library for Nearest Neighbor Search), http://www.cs.ubc.ca/research/flann

[6] Nearest Neighbor Search – Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Nearest_neighbor_search

[7] Octree - Wikipedia, the free encyclopedia http://en.wikipedia.org/wiki/Octree

[8] Structure of the Hadoop Distributed File System – Blogger, http://fbe-big-data.blogspot.com/2013/01/structure-of-hadoop-distributed.html

[9] Overview of Namenode & Datanode – Blogger http://hadooptutor.blogspot.com/2013/08/namenode-secondary-namenode-and.html

[10] Typical Hadoop Cluster – Hortonworks. http://docs.hortonworks.com/HDPDocuments/HDP1/HDP-1.3.2/bk_getting-started-guide/content/ch_hdp1_getting_started_chp3.html

[11] Hadoop MapReduce – TutorialsPoint http://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm

[12] Dean, Jeffrey., Ghemawat, Sanjay (2004). MapReduce: Simplified Data Processing on Large Clusters. Google Inc. OSDI, 2004

[13] Zhao, Jerry., Pjesivac-Grbovic, Jelena (2009). MapReduce: The Programming Model and Practice. SIGMETRICS/Performance 2009, Google Inc.

[14] Anatomy of a MapReduce Job in Apache Hadoop – Edureka, http://www.edureka.co/blog/anatomy-of-a-mapreduce-job-in-apache-hadoop/

[15] Apache Hadoop NextGen MapReduce (YARN) – The Apache Software Foundation, http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[16] History Server REST API's. - The Apache Software Foundation, https://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/HistoryServerRest.html

[17] Awadallah, Amr (2011). Introducing Apache Hadoop: The Modern Day Operating System. Stanford EE380 Computer Systems Colloquium 2011, Cloudera Inc.

[18] Singh, Dilpreet., Reddy, Chandan K. (2014). A Survey on Platforms for Big Data Analytics. *Journal of Big Data*, 2014

[19] Jimmy Linn, Chris Dyer. Data-Intensive Text Processing with MapReduce. University of Maryland, College Park

[20] Iyla Katsov. MapReduce Patterns, Algorithms and Use Cases. Highly Scalable Blog, https://highlyscalable.wordpress.com/2012/02/01/mapreduce-patterns/

[21] Ricky Ho. Pragmatic Programming Techniques. Blogger, http://horicky.blogspot.com/2010/08/designing-algorithmis-for-map-reduce.html

[22] Donald Miner, Adam Shook. MapReduce Design Patterns. O' Reilly Media Inc.

[23] Fatemeh Alsadat Estiri. 3D Object Detection and Tracking Based on Point Cloud Library Special Application In Pallet Picking for Autonomous Mobile Machines. Tampere University of Technology, 2014.

[24] Hadoop Streaming – The Apache Software Foundation, http://hadoop.apache.org/docs/r1.2.1/streaming.html

[25] Digital 210 King – Autodesk Research, http://www.digital210king.org/downloads.php

[26] Point Cloud Modelling, http://www.pointcloudmodeling.com/samples.htm

[27] XYZ to PCD Conversion Tool – GitHub, https://github.com/PointCloudLibrary/pcl/blob/master/tools/xyz2pcd.cpp

[28] Online LIDAR Point Cloud Viewer, http://lidarview.com/

[29] Graph Individual (x, y) Points – WebMath, www.webmath.com/gpoints.html

[30] Maximizing Hadoop Performance & Storage Capacity – Exar, http://www.exar.com/uploadedfiles/products/data_compression/mwp-0002_a01_maximizinghadoopperfandstoragecapacitywithaltrahd.pdf