Point cloud data management benchmark: Oracle, PostgreSQL, MonetDB and LAStools

Oscar Martinez-Rubi, Peter van Oosterom, Romulo Gonçalves, Theo Tijssen
(TU Delft and Netherlands eScience Center)

Management of massive point cloud data: wet and dry (2), Delft, 8 December 2015
Content overview

0. Background

1. Conceptual benchmark

2. Executable benchmark

3. Conclusion and future work
NL eScience Point cloud project

- TU Delft:
  1. GIS technology
  2. TU Delft, Library, contact with research & education users, dissemination & disclosure of point cloud data
  3. 3TU.Datacentrum, long-term provision of ICT-infra
  4. TU Delft Shared Service Center ICT, storage facilities

- NL eScience Center, designing and building ICT infrastructure

- Oracle Spatial, New England Development Centre (USA), improving existing software
- Rijkswaterstaat, data owner (and in-house applications)
- Fugro, point cloud data producer
- CWI, Amsterdam, MonetDB group
User requirements

- report user requirements, based on structured interviews with
  - Government community: RWS (Ministry)
  - Commercial community: Fugro (company)
  - Scientific community: TU Delft Library

- report at MPC public website [http://pointclouds.nl](http://pointclouds.nl)

- basis for conceptual benchmark, with tests for functionality, classified by importance (based on user requirements and Oracle experience)
Point cloud data

- Not new, but growing rapidly (with increasing data management problems)
- Some producing technologies:
  - Laser scanning (terrestrial, airborne)
  - Multi-beam echo sounding
  - Stereo photogrammetry (point matching)

- Results are huge data sets, very detailed (precise), information rich

- AHN (2009):
  1 point per 16 m²
- AHN2 (2014):
  10 points per 1 m²

Many Terabytes
Case AHN2 (open data in NL)

• TU Delft Library: distribution point for (geo-)data, including AHN2
• Users include:
  • Architecture (Urbanism, Landscape architecture),
  • Civil Engineering and Geosciences (Water Management, Geo-engineering)
  • Aerospace Engineering (Mathematical Geodesy & Positioning)
  • Electrical Engineering, Mathematics and Computer Science (Computer Graphics & Visualisation)
  • ‘Outside’ the TU Delft (but on campus): Deltares
  • More and more students are using this data

• Challenge: how to make this big data useful to the users? (AHN2: 640,000,000,000 points with 3 cm hor./vert. accuracy)
Applications, often related to the environment

- examples:
  - flood modeling,
  - dike monitoring,
  - forest mapping,
  - generation of 3D city models, etc.

- it is expected that future data sets will feature an even higher point density (Cyclomedia announced a 35 trillion pts NL data set)

- because of a lack of (processing) tools, most of these datasets are not being used to their full potential (e.g. first converting to 0.5 m grid or 5 m grid, the data is losing potentially significant detail)
Approach

• develop infrastructure for the storage, the management, … of massive point clouds (note: no object reconstruction)

• support range of hardware platforms: normal/department servers (HP), cloud-based solution (MS Azure), Exadata (Oracle)

• scalable solution: if data sets becomes 100 times larger and/or if we get 1000 times more users (queries), it should be possible to configure based on same architecture

• generic, i.e. also support other (geo-)data and standards based, if non-existent, then propose new standard to ISO (TC211/OGC): Web Point Cloud Service (WPCS)

• also standardization at SQL level (SQL/SFS, SQL/raster, SQL/PC)?
Why a DBMS approach?

- today’s common practice: specific file format (LAS, LAZ, ZLAS,…) with specific tools (libraries) for that format

- point clouds are a bit similar to raster data: sampling nature, huge volumes, relatively static

- specific files are sub-optimal data management:
  - multi-user (access and some update)
  - scalability (not nice to process 60,000 AHN2 files)
  - integrate data (types: vector, raster, administrative)

- ‘work around’ could be developed, but that’s building own DBMS
- no reason why point cloud cannot be supported efficiently in DBMS
- perhaps ‘mix’ of both: use file (or GPU) format for the PC blocks
Content overview

0. Background

1. **Conceptual benchmark**

2. Executable benchmark

3. Conclusion and future work
Benchmark organization

• mini-benchmark, small subset of data
  (20 million points) + limited functionality
  → get experience with benchmarking, platforms
  → first setting for tuning parameters: block size, compression

• medium-benchmark, larger subset
  (20 billion points) + more functionality
  → more serious testing, first feeling for scalability
  → more and different types of queries (e.g. nearest neighbour)

• full-benchmark, full AHN2 data set
  (640 billion points) + yet more functionality
  → LoD (multi-scale), multi-user test

• scaled-up benchmark, replicated data set
  (20 trillion points) → stress test
Test data: AHN2
(subsets)

<table>
<thead>
<tr>
<th>Name</th>
<th>Points</th>
<th>LAS files</th>
<th>Disk size [GB]</th>
<th>Area [km²]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20M</td>
<td>20,165,862</td>
<td>1</td>
<td>0.4</td>
<td>1.25</td>
<td>TU Delft campus</td>
</tr>
<tr>
<td>210M</td>
<td>210,631,597</td>
<td>16</td>
<td>4.0</td>
<td>11.25</td>
<td>Major part of Delft city</td>
</tr>
<tr>
<td>2201M</td>
<td>2,201,135,689</td>
<td>153</td>
<td>42.0</td>
<td>125</td>
<td>City of Delft and surroundings</td>
</tr>
<tr>
<td>23090M</td>
<td>23,090,482,455</td>
<td>1,492</td>
<td>440.4</td>
<td>2,000</td>
<td>Major part of Zuid-Holland province</td>
</tr>
<tr>
<td>639478M</td>
<td>639,478,217,460</td>
<td>60,185</td>
<td>11,644.4</td>
<td>40,000</td>
<td>The Netherlands</td>
</tr>
</tbody>
</table>
HP DL380p Gen8

‘Normal’ server hardware configuration:

- HP DL380p Gen8 server
  1. 2 x 8-core Intel Xeon processors (32 threads), E5-2690 at 2.9 GHz
  2. 128 GB main memory (DDR3)
  3. Linux RHEL 6.5 operating system

- Disk storage – direct attached
  1. 400 GB SSD (internal)
  2. 6 TB SAS 15K rpm in RAID 5 configuration (internal)
  3. 2 x 41 TB SATA 7200 rpm in RAID-5 configuration
     (external in 4U rack 'Yotta-III' box, 24 disks)
Exadata X4-2: Oracle SUN hardware for Oracle database software

- database Grid: multiple Intel cores, computations
  Eight, quarter, half, full rack with resp. 24, 48, 96, 192 cores
- storage Servers: multiple Intel cores, massive parallel smart scans
  (predicate filtering, less data transfer, better performance)
- hybrid columnar compression (HCC): query and archive modes
Content overview

0. Background

1. Conceptual benchmark

2. Executable benchmark

3. Conclusion and future work
First executable mini-benchmark

- load small AHN2 dataset (one of the 60,000 LAS files) in:
  1. Oracle PointCloud
  2. Oracle flat (1 x,y,z attribute per row, btree index on x,y)
  3. PostgreSQL PointCloud
  4. PostgreSQL flat (1 2D point + z attribute per row, spatial index)
  5. MonetDB flat (1 x,y,z attribute per “row”, no index)
  6. LASTools (file, no database, tools from Rapidlasso, Martin Isenburg)

- no compression, PC block size 5000, one thread, xyz only
- input 20,165,862 XYZ points (LAS 385 MB, LAZ 37 MB)
Oracle 12c PointCloud (SDO_PC)

- point cloud metadata in SDO_PC object
- point cloud data in SDO_PC_BLK object (block in BLOB)
- loading: text file X,Y,Z,… using bulk loader (from LAS files) and use function SDO_PC_PKG.INIT and SDO_PC_PKG.CREATE_PC procedure (time consuming)
- block size 5000 points
- various compression options (initially not used)

- no white areas
- non-overlapping blocks
- 4037 blocks:
  - 4021 with 5000 points
  - some with 4982 - 4999 points
  - some others with 2501 - 2502 points
PostgreSQL PointCloud

- use PointCloud extension by Paul Ramsey
  https://github.com/pramsey/pointcloud
- also PostGIS extension (query)
- loading LAS(Z) with PDAL pcpipeline
- block size 5000 points
- spatial GIST index for the blocks

- white areas
- 4034 blocks:
  - 3930 blocks with 4999 points
  - 104 blocks with 4998 points
MonetDB

• MonetDB: open source column-store DBMS developed by Centrum Wiskunde & Informatica (CWI), the Netherlands

• MonetDB/GIS: OGC simple feature extension to MonetDB/SQL

• no support for blocked model → only flat model tested

• no need to specify index (will be created on-the-fly when needed by first query…)
LASTools (use licensed/paid version)

- programming API LASlib (with LASzip DLL) that implements reading and writing LiDAR points from/to ASPRS LAS format ([http://lastools.org/](http://lastools.org/) or [http://rapidlasso.com/](http://rapidlasso.com/))
- LASTools: collection of tools for processing LAS or LAZ files; e.g. lassort.exe (z-orders), lasclip.exe (clip with polygon), lasthin.exe (thinning), las2tin.exe (triangulate into TIN), las2dem.exe (rasterizes into DEM), las2iso.exe (contouring), lasview.exe (OpenGL viewer), lasindex.exe (index for speed-up), …

- command: `lasindex [LAS File path]`
  create LAX file per LAS file with spatial indexing info
- some tools only work in Windows, for Linux Wine ([http://www.winehq.org](http://www.winehq.org))
- note: file based solution, inefficient for large number of files; AHN2 data sets consists of over 60,000 LAZ (and LAX) files
Esri’s LiDAR file format: ZLAS

- Esri LAS Optimizer/Compressor into ZLAS format
- standalone executable, ArcGIS not required
- same executable EzLAS.exe for compression and decompression

- compression a bit disappointing: from 385 MB to 42 MB (factor 9) compared to LAZ 36 MB (factor 10)
- perhaps the 'use' performance is better (in Esri tools)

- not further tested in benchmark
From **mini- to medium-benchmark**: load (index) times and sizes

- **p**=Postgres, **o**=Oracle, **m**=MonetDB, **lt**=LAStools
- **f**=flat model, **b**=blocked model
- 20, 210, 2201, 23090M = millions of points

<table>
<thead>
<tr>
<th>Approach</th>
<th>Time[s]</th>
<th>Size[MB]</th>
<th>Points</th>
<th>Points/s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Init.</td>
<td>Load</td>
<td>Close</td>
</tr>
<tr>
<td>pf20M</td>
<td>44.07</td>
<td>2.00</td>
<td>13.86</td>
<td>28.21</td>
</tr>
<tr>
<td>pf210M</td>
<td>771.63</td>
<td>2.09</td>
<td>71.44</td>
<td>698.10</td>
</tr>
<tr>
<td>pf2201M</td>
<td>9897.37</td>
<td>1.15</td>
<td>722.91</td>
<td>9173.31</td>
</tr>
<tr>
<td>pf23090M</td>
<td>95014.05</td>
<td>3.64</td>
<td>8745.90</td>
<td>86264.51</td>
</tr>
<tr>
<td>of20M</td>
<td>128.43</td>
<td>0.51</td>
<td>123.92</td>
<td>4.00</td>
</tr>
<tr>
<td>of210M</td>
<td>275.72</td>
<td>0.32</td>
<td>230.01</td>
<td>45.39</td>
</tr>
<tr>
<td>of2201M</td>
<td>1805.68</td>
<td>0.31</td>
<td>1228.81</td>
<td>576.56</td>
</tr>
<tr>
<td>of23090M</td>
<td>15825.53</td>
<td>0.14</td>
<td>9226.37</td>
<td>6599.02</td>
</tr>
<tr>
<td>mf20M</td>
<td>7.15</td>
<td>1.14</td>
<td>3.70</td>
<td>2.31</td>
</tr>
<tr>
<td>mf210M</td>
<td>29.15</td>
<td>1.13</td>
<td>16.63</td>
<td>11.39</td>
</tr>
<tr>
<td>mf2201M</td>
<td>304.14</td>
<td>4.91</td>
<td>198.76</td>
<td>100.47</td>
</tr>
<tr>
<td>mf23090M</td>
<td>8490.90</td>
<td>0.93</td>
<td>5466.99</td>
<td>3022.98</td>
</tr>
<tr>
<td>pb20M</td>
<td>153.41</td>
<td>22.03</td>
<td>130.53</td>
<td>0.85</td>
</tr>
<tr>
<td>pb210M</td>
<td>129.14</td>
<td>0.81</td>
<td>121.00</td>
<td>7.33</td>
</tr>
<tr>
<td>pb2201M</td>
<td>754.22</td>
<td>0.86</td>
<td>687.49</td>
<td>65.87</td>
</tr>
<tr>
<td>pb23090M</td>
<td>12263.05</td>
<td>0.87</td>
<td>7450.10</td>
<td>4812.08</td>
</tr>
<tr>
<td>ob20M</td>
<td>296.22</td>
<td>0.35</td>
<td>228.17</td>
<td>67.70</td>
</tr>
<tr>
<td>ob210M</td>
<td>1246.87</td>
<td>0.25</td>
<td>557.70</td>
<td>688.92</td>
</tr>
<tr>
<td>ob2201M</td>
<td>16737.02</td>
<td>0.86</td>
<td>7613.39</td>
<td>9122.77</td>
</tr>
<tr>
<td>ob23090M</td>
<td>192612.07</td>
<td>0.31</td>
<td>96148.06</td>
<td>96463.70</td>
</tr>
<tr>
<td>lt20M</td>
<td>9.49</td>
<td>0.03</td>
<td>9.46</td>
<td>0.00</td>
</tr>
<tr>
<td>lt210M</td>
<td>30.07</td>
<td>0.02</td>
<td>28.68</td>
<td>1.37</td>
</tr>
<tr>
<td>lt2201M</td>
<td>218.06</td>
<td>0.02</td>
<td>216.18</td>
<td>1.86</td>
</tr>
<tr>
<td>lt23090M</td>
<td>2129.30</td>
<td>0.03</td>
<td>2116.64</td>
<td>12.63</td>
</tr>
</tbody>
</table>
Query geometries (mini-benchmark)

1. small rectangle, axis aligned, 51 x 53 m
2. large rectangle, axis aligned, 222 x 223 m
3. small circle at (85365 446594), radius 20 m
4. large circle at (85759 447028), radius 115 m
5. simple polygon, 9 points
6. complex polygon, 792 points, 1 hole
7. long narrow diagonal rectangle
SQL Query syntax (geometry 1)

• PostgreSQL PointCloud: CREATE TABLE query_res_1 AS SELECT PC_Explode(PC_Intersection(pa,geom))::geometry FROM patches pa, query_polygons WHERE pc_intersects(pa,geom) AND query_polygons.id = 1; note, actually points have been converted to separate x,y,z values

• Oracle PointCloud: CREATE TABLE query_res_1 AS SELECT * FROM table (sdo_pc_pkg.clip_pc(SDO_PC_object, (SELECT geom FROM query_polygons WHERE id = 1), NULL, NULL, NULL, NULL)); note SDO_PC_PKG.CLIP_PC function will return SDO_PC_BLK objects, actually have been converted via geometry (multipoint) with SDO_PC_PKG.TO_GEOMETRY function to separate x,y,z values

• LASTools: lasclip.exe [LAZ File] -poly query1.shp -verbose -o query1.laz
Queries: returned points + times
(note flat model: increasing times)

- Scalability flat model: an issue

<table>
<thead>
<tr>
<th>Approach</th>
<th>Number of points</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>pf20M</td>
<td>74947</td>
<td>0.35</td>
</tr>
<tr>
<td>pf210M</td>
<td>74947</td>
<td>0.42</td>
</tr>
<tr>
<td>pf2201M</td>
<td>74947</td>
<td>4.92</td>
</tr>
<tr>
<td>pf23090M</td>
<td>74947</td>
<td>5.17</td>
</tr>
<tr>
<td>of20M</td>
<td>74872</td>
<td>0.24</td>
</tr>
<tr>
<td>of210M</td>
<td>74872</td>
<td>0.45</td>
</tr>
<tr>
<td>of2201M</td>
<td>74872</td>
<td>1.47</td>
</tr>
<tr>
<td>of23090M</td>
<td>74872</td>
<td>1.25</td>
</tr>
<tr>
<td>mf20M</td>
<td>74872</td>
<td>0.06</td>
</tr>
<tr>
<td>mf210M</td>
<td>74872</td>
<td>0.13</td>
</tr>
<tr>
<td>mf2201M</td>
<td>74872</td>
<td>0.64</td>
</tr>
<tr>
<td>mf23090M</td>
<td>74872</td>
<td>7.21</td>
</tr>
<tr>
<td>pb20M</td>
<td>74947</td>
<td>0.32</td>
</tr>
<tr>
<td>pb210M</td>
<td>74947</td>
<td>0.32</td>
</tr>
<tr>
<td>pb2201M</td>
<td>74947</td>
<td>0.31</td>
</tr>
<tr>
<td>pb23090M</td>
<td>74947</td>
<td>0.32</td>
</tr>
<tr>
<td>ob20M</td>
<td>74947</td>
<td>0.41</td>
</tr>
<tr>
<td>ob210M</td>
<td>74947</td>
<td>0.38</td>
</tr>
<tr>
<td>ob2201M</td>
<td>74947</td>
<td>0.39</td>
</tr>
<tr>
<td>ob23090M</td>
<td>74947</td>
<td>0.40</td>
</tr>
<tr>
<td>lt20M</td>
<td>74840</td>
<td>0.04</td>
</tr>
<tr>
<td>lt210M</td>
<td>74840</td>
<td>0.06</td>
</tr>
<tr>
<td>lt2201M</td>
<td>74840</td>
<td>0.06</td>
</tr>
<tr>
<td>lt23090M</td>
<td>74840</td>
<td>0.05</td>
</tr>
</tbody>
</table>
Full AHN2 benchmark: loading 640 B (Exadata: different hardware)

<table>
<thead>
<tr>
<th>system</th>
<th>Total load time [hours]</th>
<th>Total size [TB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAStools LAS</td>
<td>22:54</td>
<td>12.18</td>
</tr>
<tr>
<td>LAStools LAZ</td>
<td>19:41</td>
<td>1.66</td>
</tr>
<tr>
<td><strong>Oracle Exadata</strong></td>
<td>4:39</td>
<td>2.24</td>
</tr>
<tr>
<td>Oracle/ PDAL</td>
<td>33:53</td>
<td>2.07</td>
</tr>
<tr>
<td>MonetDB</td>
<td>17:21</td>
<td>15.0</td>
</tr>
</tbody>
</table>
Query performance full AHN2
(all on same hardware)

- LAS and LAZ solutions with database wrapper for the 60,000 files
- LAS (uncompressed) fastest in query
- LAZ and Oracle/PDAL in same league
- LAS and LAZ do not support query with holes (query 6)

<table>
<thead>
<tr>
<th>Query</th>
<th>LAStools/LAS</th>
<th>LAStools/LAZ</th>
<th>Oracle/PDAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#points</td>
<td>Time[s]</td>
<td>#pts/s</td>
</tr>
<tr>
<td>1</td>
<td>74850</td>
<td>0.03</td>
<td>2495000</td>
</tr>
<tr>
<td>2</td>
<td>717959</td>
<td>0.08</td>
<td>8974488</td>
</tr>
<tr>
<td>3</td>
<td>34691</td>
<td>0.02</td>
<td>1734550</td>
</tr>
<tr>
<td>4</td>
<td>563037</td>
<td>0.09</td>
<td>6255967</td>
</tr>
<tr>
<td>5</td>
<td>182861</td>
<td>0.15</td>
<td>1219073</td>
</tr>
<tr>
<td>6</td>
<td>460096</td>
<td>1.40</td>
<td>328640</td>
</tr>
<tr>
<td>7</td>
<td>45811</td>
<td>0.24</td>
<td>190879</td>
</tr>
</tbody>
</table>
Content overview

0. Background

1. Conceptual benchmark

2. Executable benchmark

3. Conclusion and future work
Conclusion

• Designed and executed point cloud benchmark
• Influenced point cloud data management developers (systems improved, sometimes dramatically, orders of magnitude)
• Developed an interactive 3D point cloud webservice (and viewer)

• All code developed open source (majority in SQL + Python)
  • https://github.com/NLeSC/pointcloud-benchmark
  • https://github.com/NLeSC/Massive-PotreeConverter

• A lot of future work:
  • Multi-user testing (based on collected use patterns)
  • Discrete LoD testing (perspective views)
  • Investigate continuous LoD
  • Add more countries (“OpenPointCloudMap”, with upload facility)
  \[→\text{nD-PointCloud (submitted H2020 FET Open): } \text{http://nd-pc.org}\]
Acknowledgements

- The massive point cloud research is supported by Netherlands eScience Center, the Netherlands Organisation for Scientific Research (NWO) (project code: 027.012.101)
Interested?

• More reading in Springer paper:
  van Oosterom, P., Martinez-Rubi, O., Ivanova, M., Horhammer, M., Geringer, D., Ravada, S., Tijssen, T., Kodde, M., Gonçalves, R.
  Massive point cloud data management: Design, implementation and execution of a point cloud benchmark

• Join OGC’s Point Cloud DWG
  http://www.opengeospatial.org/projects/groups/pointclouddwg

• Try our 640,000,000,000 points web-based 3D point cloud viewer at http://ahn2.pointclouds.nl (comments welcome)