Voxel Data Management System (VDMS) Report

User requirements and conceptual benchmark

The user requirements have been listed in three categories: selections, calculations and analyses.

The user requirements have been listed in a number of categories: functionality within the system (selections, calculations and analyses), presentation functionality (access, view, navigate) and download functionality. Some functionality was examined in more detail (attributes).

Selections

- Selection: rectangle
- Selection: line + buffer
- Selection: polygon
- Selection: attributes

Calculations

Many researchers remarked that they prefer to make calculations using their own software. If possible they would like to be able to preview the results of calculations within the system before downloading the points to their local system and doing their own re-calculations. The most important calculations that the researchers would like to be able to make are: (1) calculation of statistical values, like the average height of an area, lowest point, highest point and (2) voxels density.

- Calculations: statistic values
- Calculations: point clouds density

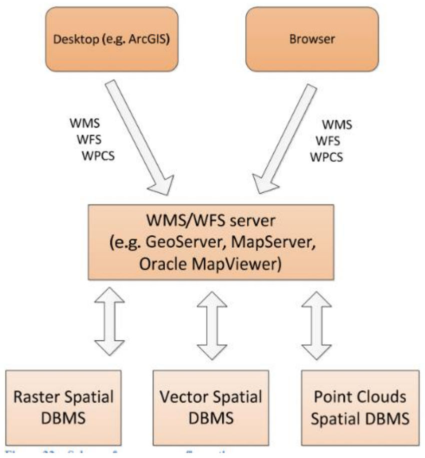
Analyses

Researchers would like to compare point clouds with data coming from other sources of the same area.

- Analysis: spatial analyses with other tables
- Analysis: pattern (nearest neighbor analysis)

Access

Most researchers (84 %) want to access the new point clouds spatial DBMS through a web service, either directly through their browser or by letting their desktop software (e.g. ArcGIS) connect to the DBMS.



View
Like with access, most researchers want to view the selected data in a web application using one of the standard browsers, or by means of a desktop application.

Navigate
Most researchers want to navigate the point cloud data using traditional techniques like zooming and panning.

Download form and format
The raster data is still the most popular format but it is reassuring to find that a lot of researchers also would like to download the point cloud data, if available. A number of download formats have been mentioned. For raster data the researchers would like to use image or publication formats (jpg, pdf) or GeoTIFF and NetCDF format [Website: NetCDF, Website: OpenDAP]. For point clouds downloads in LAS an LAZ formats are the current standard.

Co-ordinate systems
Different CPS transform.

Functionality in the various categories
http://www.pointclouds.nl/index.html

Benchmark stages

Dataset name	Benchmark	#Voxels	#Files	Format	Disk size (GB)	Description
Village	Small	22,333,289	1	xyz	0.32	'UNSW Village' building
building	Medium	241,613,693	52	xyz	3.52	All buildings in UNSW lower campus
campus	Large	942,878,048	54	xyz	13.62	All buildings, terrain and tree in UNSW lower campus

General VDMS description

Name	Software	Software description	Type
pyf	Python/PySpark	Python and its libraries	File-based
pf	PostgreSQL	Open-source RDBMS	Flat table
pm	PostgreSQL/PostGIS	Open-source RDBMS	Multipoint geometry
pp	PostgreSQL/PostGIS	Open-source RDBMS	PcPatch
mf	MongoDB	Document-oriented NoSQL	Flat table
mm	MongoDB	Document-oriented NoSQL	Multipoint geometry

Management of voxels using files

Binary file

Feather file

Management of voxels using RDBMS

PostgreSQL

Create database

```
postgres=> CREATE DATABASE VDMS WITH ENCODING='UTF-8' OWNER=wei CONNECTION
LIMIT=100;
CREATE DATABASE
```

```
postgres=> \c VDMS
psql (11.2, server 12.3)
WARNING: psql major version 11, server major version 12.
         Some psql features might not work.
WARNING: Console code page (850) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
You are now connected to database "VDMS" as user "wei".
```

Create tables

Management of voxels using NoSQL

MongoDB

Create table and index

```
PS H:\> cd 'D:\Program Files\MongoDB\Server\4.4\bin'

PS D:\Program Files\MongoDB\Server\4.4\bin> .\mongo.exe

> show dbs

> use voxelDB

> db.createCollection("voxelpt")

> db.createCollection("voxelmpt")

> show collections
```

Import data

GeoJSON – Point

```
{

“category”: “building”,

“name”: “Red Centre-H13”,

“ifc”: “IfcDoor”,

"geom": {

      "type": "Point",

      "coordinates": [125.6, 10.1]

  }

}
```

GeoJSON – MultiPoint

```
{

“category”: “building”,

“name”: “Red Centre-H13”,

“ifc”: “IfcDoor”,

"geom": {

      type: "MultiPoint",

  coordinates: [

    [ -73.9580, 40.8003 ],
    [ -73.9498, 40.7968 ],
    [ -73.9737, 40.7648 ],
    [ -73.9814, 40.7681 ]
  ]  }
}
```

Step-1: EPSG: 28356 to EPSG:4326

EPSG:28356	EPSG:4326
• WGS84 Bounds: 150.0000, -37.8000, 156.0000, -21.7000 • Projected Bounds: 189586.6272, 5812134.5296, 810413.3728, 7597371.5494 • Area: Australia - 150°E to 156°E	• WGS84 Bounds: -180.0000, -90.0000, 180.0000, 90.0000 • Projected Bounds: -180.0000, -90.0000, 180.0000, 90.0000 • Area: World

Solution-1: Use ST_Transform() to convert EPSG: 28356 to EPSG:4326 in SQL script, and output it as a csv file.

Solution-2: Use GDAL API to code a program to do such transformation in C++.

Rasdaman

Background

Traditional databases do not support the information category of large multidimensional arrays, while rasdaman technology offers distinct array management features whose conceptual model supports arrays of any number of dimensions and over virtually any cell ("***pixel***", "***voxel***") type. The rasdaman query language, ***rasql***, is crafted along standard SQL and gives high-level, declarative access to any kind of raster data. Its architecture principle of tile stream processing, together with highly effective optimizations, has proven scalable into multi-Terabyte object sizes.

Array data model. Arrays are determined by their extent ("domain") and their cell ("pixel", "voxel"). Over such typed arrays, collections (similar to table in RDBMS) are built. Collections have two columns (attributes), a system-maintained object identifier (OID) and the array itself (i.e., array is a new attribute type). This allows to conveniently embed arrays into relational modeling: foreign keys in conventional tables allow to reference particular array objects, in connection with a domain specification even parts of arrays.

On server side, arrays are stored inside a standard database. To this end, arrays are partitioned into subarrays called **tiles**; each such tile goes into a BLOB (binary large object) in a relational table. This allows conventional relational database systems to maintain arrays of unlimited size. Besides, a spatial index allows to quickly locate the tiles required for determining the tile set addressed by a query.

Query language. The rasdaman query language, rasql, offers raster processing formulated through expressions over raster operations in the style of SQL. Rasql is a full query language, supporting select, insert, update, and delete. Additionally, the concept of a partial update is introduced which allows to selectively update parts of an array.

OGC geo standards support. Rasdaman implements the Open Geospatial Consortium standards for gridded coverages, i.e., multi-dimensional raster data. It offers spatio-temporal access and analytics through APIs based on the OGC data standard *Coverage Implementation Schema* (CIS) and the OGC service standards *Web Map Service* (WMS), *Web Coverage Service* (WCS), and *Web Coverage Processing Service* (WCPS).

Installation

Due to Rasdaman only tested on Linux OS, we choose Virtual Machine + Ubuntu 18.04 (https://releases.ubuntu.com/bionic/) as basic experimental environment. We get preconfectioned packages for installing DEB packages on Debian / Ubuntu; this is the recommended way - among others because the package manager will be able to manage the installation.

Note that, the rasdaman engine in the packages uses embedded **SQLite** for managing its array metadata, and geo service component, petascope, currently still relies on a **PostgreSQL** database.

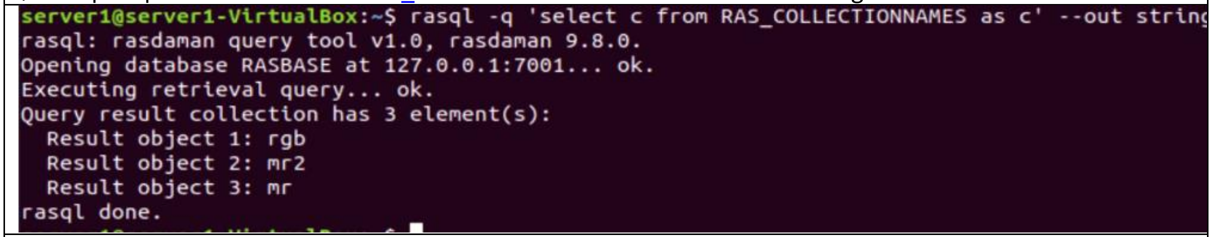
Once the rasdaman installation has been accomplished. We can customize a rasdaman script by updating environment variables, and then we can use rasdaman service script to start/stop rasdaman.

Installation

```
$ wget -O - https://download.rasdaman.org/packages/rasdaman.gpg | sudo apt-
key add -
$ echo "deb [arch=amd64] https://download.rasdaman.org/packages/deb bionic stable
" \ | sudo tee /etc/apt/sources.list.d/rasdaman.list
$ sudo apt-get update
$ sudo apt-get install rasdaman
$ source /etc/profile.d/rasdaman.sh
```

Sample query

```
$ rasql -q 'select c from RAS_COLLECTIONNAMES as c' --out string
```



```
server1@server1-VirtualBox:~$ rasql -q 'select c from RAS_COLLECTIONNAMES as c' --out strin
rasql: rasdaman query tool v1.0, rasdaman 9.8.0.
Opening database RASBASE at 127.0.0.1:7001... ok.
Executing retrieval query... ok.
Query result collection has 3 element(s):
  Result object 1: rgb
  Result object 2: mr2
  Result object 3: mr
rasql done.
```

Rasdaman service

```
# rasdaman installation directory
RMANHOME=/opt/rasdaman
# local user running the rasdaman server
RMANUSER=rasdaman
# runuser, or sudo for older OS
RUNUSER=runuser
# login credentials for non-interactive rasdaman start/stop
RASLOGIN=rasadmin:d293a15562d3e70b6fdc5ee452eaed40
# port on which clients connect to rasdaman
RASMGR_PORT=7001
# options to be passed on to start_rasdaman.sh
START_RASDAMAN_OPTS="-p $RASMGR_PORT"
# options to be passed on to stop_rasdaman.sh
STOP_RASDAMAN_OPTS="-p $RASMGR_PORT"
```

```
$ service rasdaman start
$ service rasdaman stop
$ service rasdaman status
```

To clarify Rasdaman operation, we briefly check the directory structure (the default installation directory is *$RMANHOME= /opt/rasdaman*):

Directory	**Description**
bin	rasdaman executables, e.g. rasql, start_rasdaman.sh, …
data	Path where the server stores array tiles as files; this directory can get big, it is recommended to make it a link to a sufficiently large disk partition.
etc	Configuration files, e.g. rasmgr.conf

Small benchmark

This section contains the description of a small benchmark focussed on the storage and management of the voxel data. We use the term "small" to reflect the fact that we use a small test dataset and a small subset of test queries. The goal of the project is to develop a system that can deal with a much larger amount of points while providing a much larger set of functionalities.

We use a small dataset which consists of 22,333,289 voxels of the building – 'UNSW Village' in the UNSW Kensington campus in Australia. Figure 1 depicts the 3D spatial representation of the building. The voxels in this dataset belong to one of the more than 50 objects of UNSW Kensington campus. The input data is provided as a XYZ file and its size is 329 MB. Each voxel has 4 dimensions which are listed in the Table 1. Note that the spatial coordinates, which are given in meters, are stored in 32 bits. Actually, they are stored as scaled integers, which can also have an offset. The scale is 0.2 which means that the coordinates are stored in meter. In this case, the X, Y, Z coordinate also has an offset of 336000, 6245250 and 20 meters, respectively.

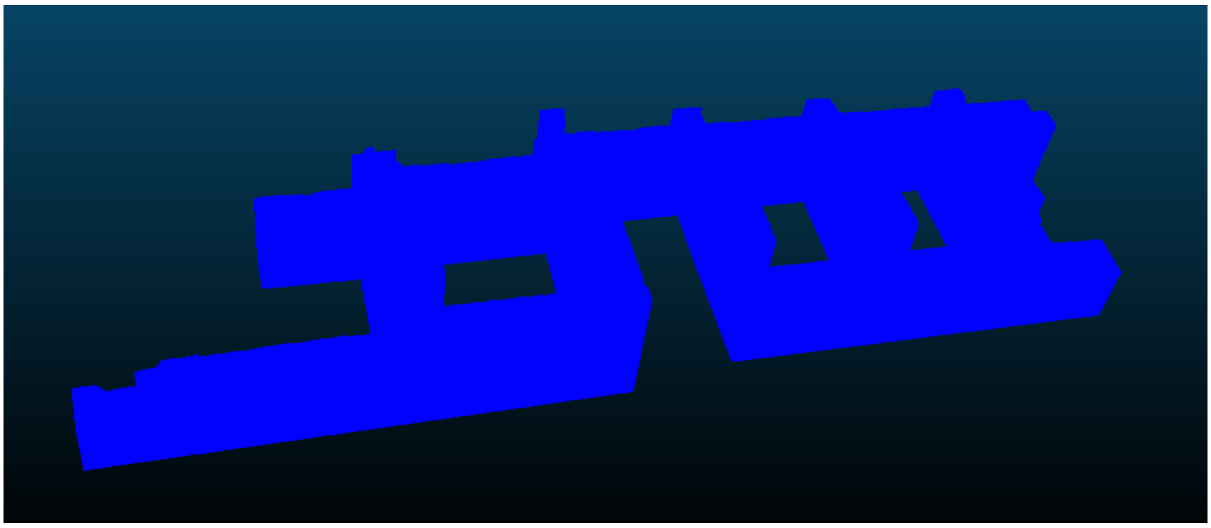


Figure 1: 3D representation of the area used in the small benchmark

Name	Size[bits]	Offset[bytes]
X	32	0
Y	32	4
Z	32	8
object ID	8	12

Table 1: Dimensions of the voxels in the input XYZ file. Only X, Y and Z contain valuable data

In order to do a fair comparison, we decided that all the approaches must meet the following requirements:

- The same input file format will be used.
- The data will be stored without compression, i.e. using the available double data types.
- In the approaches that use blocks the block size will always be the same, 5000 points.
- Only the 3 spatial dimensions will be stored in the database.
- Only one process/thread will be used during the loading and querying.
- When querying, the results will be written in a database table, one row for each queried point. We will use the SQL statement CREATE TABLE [name] AS SELECT.

In addition to the database approaches we have also implemented a "not-database" solution using the Python and its packages. In this case, since the data is directly queried from the XYZ files, many of the requirements presented above cannot be met. Thus, the comparison in this case must be done taking the latter into account.

In the next paragraphs we present the different approaches. For each implemented approach the execution of the benchmark is split in two parts:

- Loading: The test dataset is loaded into the database (this obviously do not apply in the File-based approach). The spent time, the required storage and the used resources (CPU and memory) are monitored.
- Querying: A set of (2D) queries is executed. The spent time, the returned points and the used resources (CPU and memory) are monitored.

PostgreSQL flat

In this approach we store each point in a row of a table. We make a view where we use a method to create a PostGIS 2D point (on the X and Y coordinates), in which we add a GIST index.

```
VDMS=# \i D:/Wei/VDMS/flat.sql
CREATE EXTENSION
Time: 2140.915 ms (00:02.141)
psql:D:/Wei/VDMS/flat.sql:14: NOTICE:  table "small_flat" does not exist, skipping
DROP TABLE
Time: 9.327 ms
CREATE TABLE
Time: 1.616 ms
psql:D:/Wei/VDMS/flat.sql:23: ERROR:  relation "tmp" already exists
Time: 32.247 ms
psql:D:/Wei/VDMS/flat.sql:24: ERROR:  column "objectid" of relation "tmp" already exists
Time: 28.266 ms
COPY 22333289
Time: 51860.514 ms (00:51.861)
INSERT 0 44666578
Time: 256919.182 ms (04:16.919)
DROP TABLE
Time: 389.405 ms
CREATE VIEW
Time: 33.068 ms
psql:D:/Wei/VDMS/flat.sql:34: NOTICE:  index "small_flat_xy_idx" does not exist, skipping
DROP INDEX
Time: 7.961 ms
CREATE INDEX
Time: 819734.316 ms (13:39.734)
VACUUM
Time: 971908.747 ms (16:11.909)
```

PostgreSQL Multipoint

```
VDMS=# \i D:/Wei/VDMS/flat.sql
psql:D:/Wei/VDMS/flat.sql:11: NOTICE:  extension "postgis" already exists, skipping
CREATE EXTENSION
```

```
Time: 12.597 ms
DROP TABLE
Time: 24.797 ms
CREATE TABLE
Time: 237.792 ms
CREATE TABLE
Time: 7.864 ms
COPY 22333289
Time: 85041.068 ms (01:25.041)
INSERT 0 4467
Time: 72238.351 ms (01:12.238)
DROP TABLE
Time: 287.136 ms
psql:D:/Wei/VDMS/flat.sql:43: NOTICE:  index "small_multipoint_xy_idx" does not exist, skipping
DROP INDEX
Time: 5.031 ms
CREATE INDEX
Time: 688.907 ms
VACUUM
Time: 10838.578 ms (00:10.839)
```

PostgreSQL PointCloud

We create blocks of points and we store each block in a row of a table using the pcpatch data type. The blocks are defined in the 2D (X and Y) space.

```
VDMS=# \i D:/Wei/VDMS/flat.sql
psql:D:/Wei/VDMS/flat.sql:11: NOTICE:  extension "postgis" already exists, skipping
CREATE EXTENSION
Time: 13.678 ms
psql:D:/Wei/VDMS/flat.sql:12: NOTICE:  extension "pointcloud" already exists, skipping
CREATE EXTENSION
Time: 14.911 ms
psql:D:/Wei/VDMS/flat.sql:13: NOTICE:  extension "pointcloud_postgis" already exists, skipping
CREATE EXTENSION
Time: 15.611 ms
DELETE 1
Time: 2.123 ms
INSERT 0 1
Time: 3.050 ms
DROP TABLE
Time: 15.785 ms
CREATE TABLE
Time: 288.232 ms
CREATE TABLE
Time: 10.689 ms
COPY 22333289
Time: 82029.500 ms (01:22.029)
INSERT 0 4467
Time: 50475.244 ms (00:50.475)
DROP TABLE
```

```
Time: 293.170 ms
psql:D:/Wei/VDMS/flat.sql:84: NOTICE:  index "small_pcpatch_xy_idx" does not exist, skipping
DROP INDEX
Time: 5.212 ms
CREATE INDEX
Time: 17.324 ms
VACUUM
Time: 697.753 ms
```

MongoDB flat

https://kb.objectrocket.com/mongo-db/how-to-import-a-csv-into-mongodb-327

```
> use VDMS
> db.createCollection("small_flat")
```

Medium benchmark

Large benchmark

Management of voxels under Distribution

PostgreSQL

Ref: https://cloud.tencent.com/developer/article/1563023

- **Citus** (https://www.citusdata.com/)
- pgxc && pgxl (https://www.postgres-xl.org/documentation/pgxc-ctl.html)
- **Greenplum** (https://greenplum.org/)

MongoDB

Install MongoDB Community Edition on Ubuntu

```
$ wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add –
```

```
$ echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list
```

```
$ sudo apt-get update
```

```
$ sudo apt-get install -y mongodb-org
```

```
$ ps --no-headers -o comm 1
```

Run MongoDB Community Edition

```
$ sudo systemctl start mongod
```

```
$ sudo systemctl status mongod
```

```
$ sudo systemctl stop mongod
```

```
$ sudo systemctl restart mongod
```

```
$ mongo
```

Sharding is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations. MongoDB shards data at the collection level, distributing the collection data across the shards in the cluster.

HBase

Spark

References

[1] van Oosterom, P., Martinez-Rubi, O., Ivanova, M., Horhammer, M., Geringer, D., Ravada, S., Tijssen, T., Kodde, M. and Gonçalves, R., 2015. Massive point cloud data management: Design, implementation and execution of a point cloud benchmark. *Computers & Graphics*, *49*, pp.92-125.

[2] Suijker PM, Alkemade I, Kodde MP, Nonhebel AE. User requirements Massive Point Clouds for eSciences (WP1), Technical Report, Delft University of Technology, ⟨http://repository.tudelft.nl/view/ir/uuid%3A351e0d1e-f473-4651-bf15-8f9b29b7b800/⟩; 2014

[3] Massive point clouds for eSciences, ⟨http://www.pointclouds.nl/⟩; 2014.

[4]