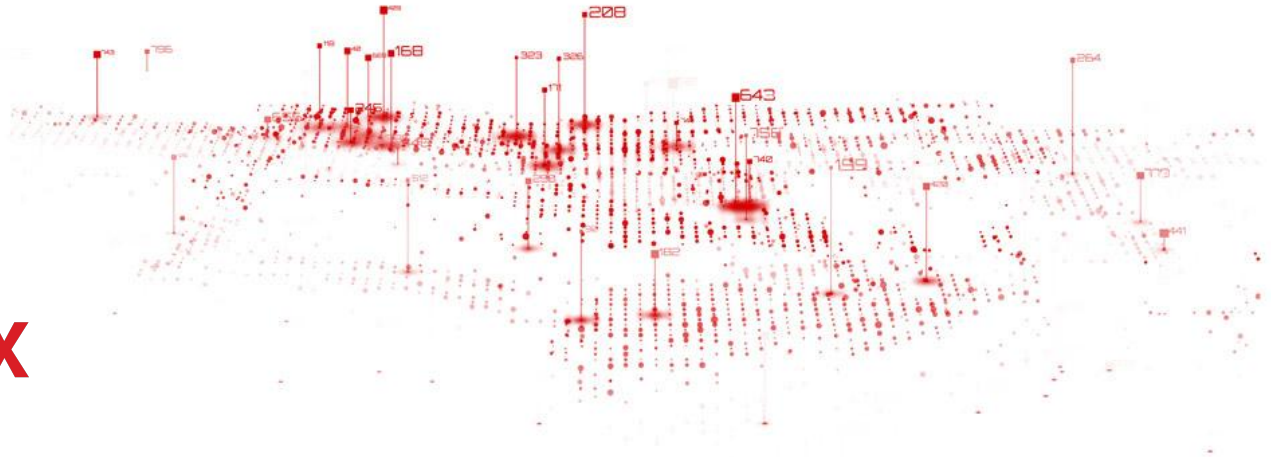
Compressing Timeseries Data

How to stop worrying and love PostgreSQL extensions

Karl Pietrzak
Lead Research Engineer
3/21/2019



twosix
LABS

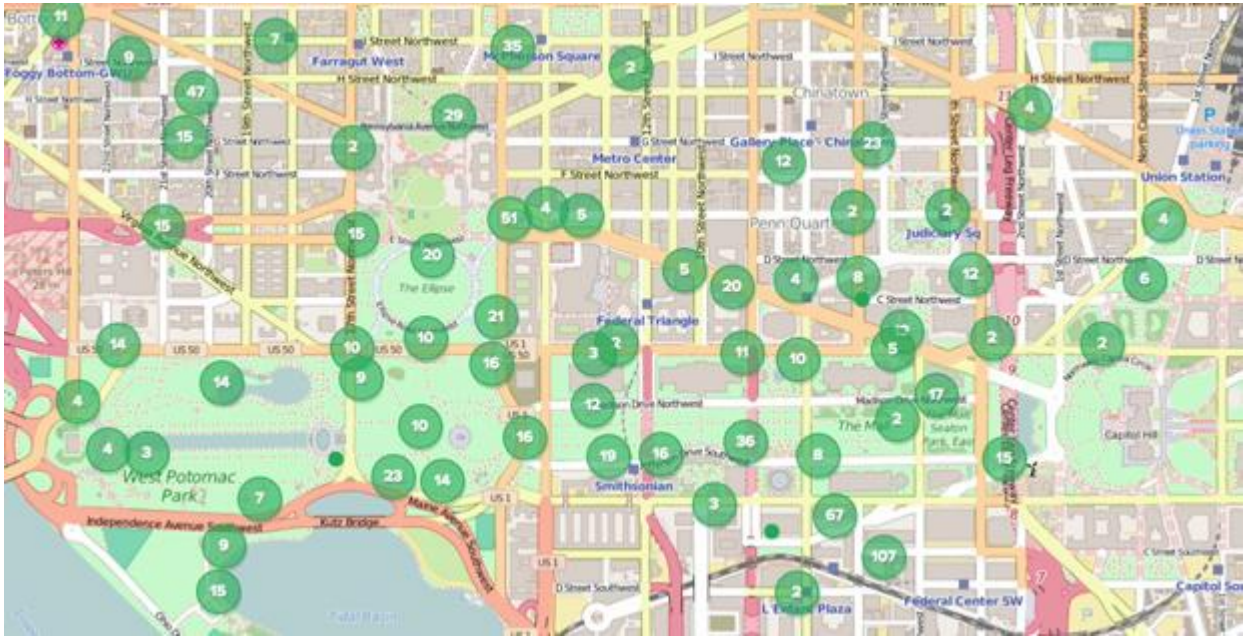
Agenda

- What we do
- Data problems we faced
- How we tried to solve them
- Results
- Other experiments

What we do

- DARPA SIGMA project
  - “Real-Time Radiological Detection and Response Platform”
  - SaaS
  - personal, mobile, and static radiation sensors
  - eventually chemical, explosives, and biological agent sensors as well

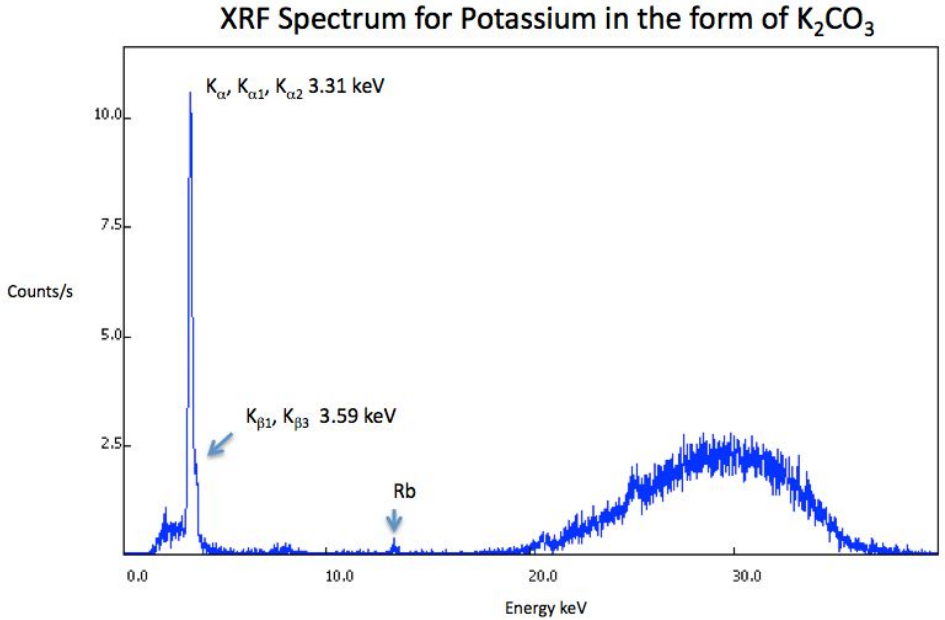
1,000 sensor deployment in DC area in 2016



https://www.darpa.mil/about-us/timeline/sigma

Data Problems We Faced

- Major IoT project
  - 100s to 1000s of sensors
  - Not just voltage and temperature
  - Full spectral readings
    - Usually 1Hz
    - Resolution of sensor varies

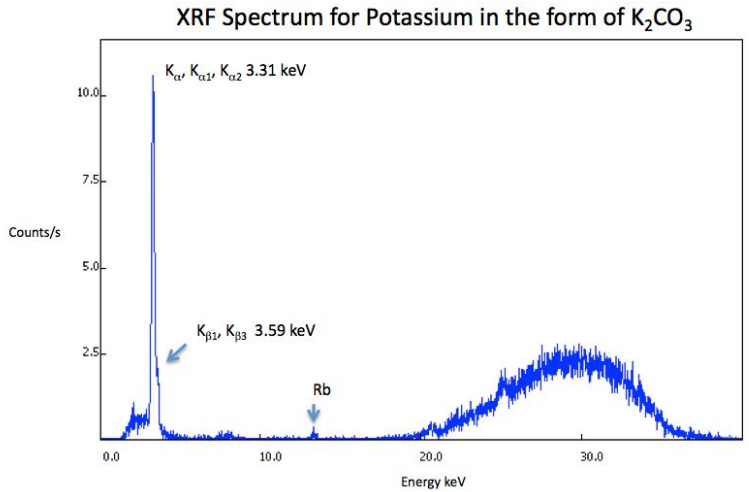


Data Problems We Faced

- Sensor ID
  - `uuid` (16 bytes)
- Time
  - `timestamptz` (8 bytes)
- Temperature
  - `real` (4 bytes)
- Battery Voltage
  - `real` (4 bytes)
- Spectrum
  - Min: 512 * `smallint` (2 bytes)
  - Max: 4096 * `smallint` (2 bytes)

=

- 16+8+8+(512*2) = 1056 bytes
- 16+8+8+(4096*2) = 8224 bytes



Data Problems We Faced

- Min: 16+8+8+(512*2) = 1056 bytes
- Max: 16+8+8+(4096*2) = 8224 bytes

×

- Once a second, 24 hours a day, 365 days a year

=

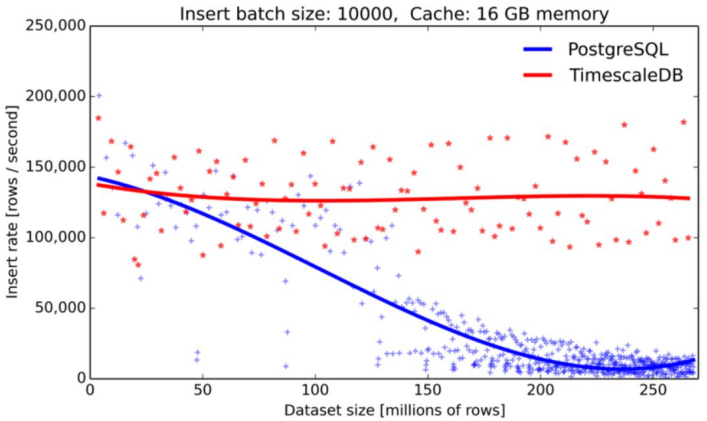
- Min: ~31GB a year
- Max: ~241GB a year
- ~31.5 billion rows a year

Data Problems We Faced

- Goal: reduce # of rows and sheer amount of data
- Data-specific
  - Battery voltage, location, spectrum, etc. don't change very much
  - Some values fall into a very narrow range
- Data-agnostic
  - `timerangetz[], float[][]` ?
  - zlib'ed `bytea` ?
  - one-byte `integer, tinyint` ?
  - partitioning, `pg_partman` ?

How we tried to solve it

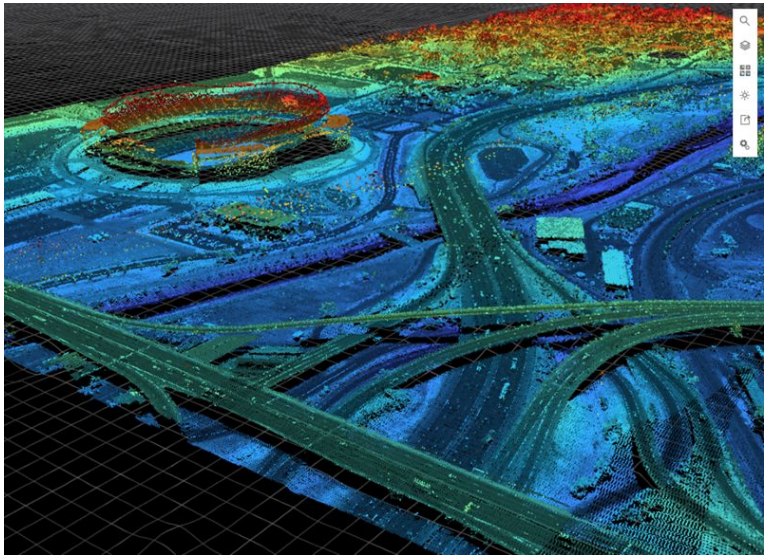
- TimescaleDB
  - open-source PostgreSQL extension
  - Manages time-based partitioning for you
  - Lots of helpful time-based functions



Insert throughput of TimescaleDB vs. PostgreSQL when performing INSERTs of 10,000-row batches.

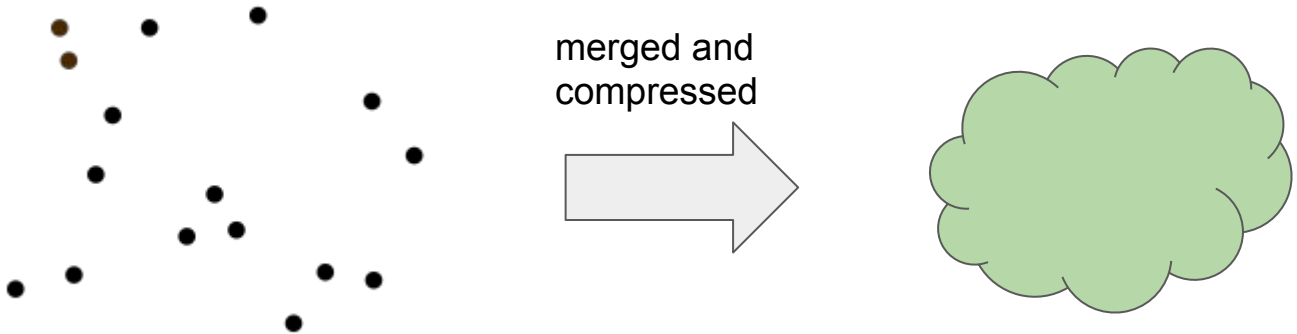
How we tried to solve it

- pgpointcloud
  - Another open-source PostgreSQL extension
  - Designed for point cloud (LIDAR) data
    - No standardized point cloud format
    - “Some data sets might contain only X/Y/Z values. Others will contain dozens of variables: X, Y, Z; intensity and return number; red, green, and blue values; return times; and many more. There is no consistency in how variables are stored: intensity might be stored in a 4-byte integer, or in a single byte; X/Y/Z might be doubles, or they might be scaled 4-byte integers.”



How we tried to solve it

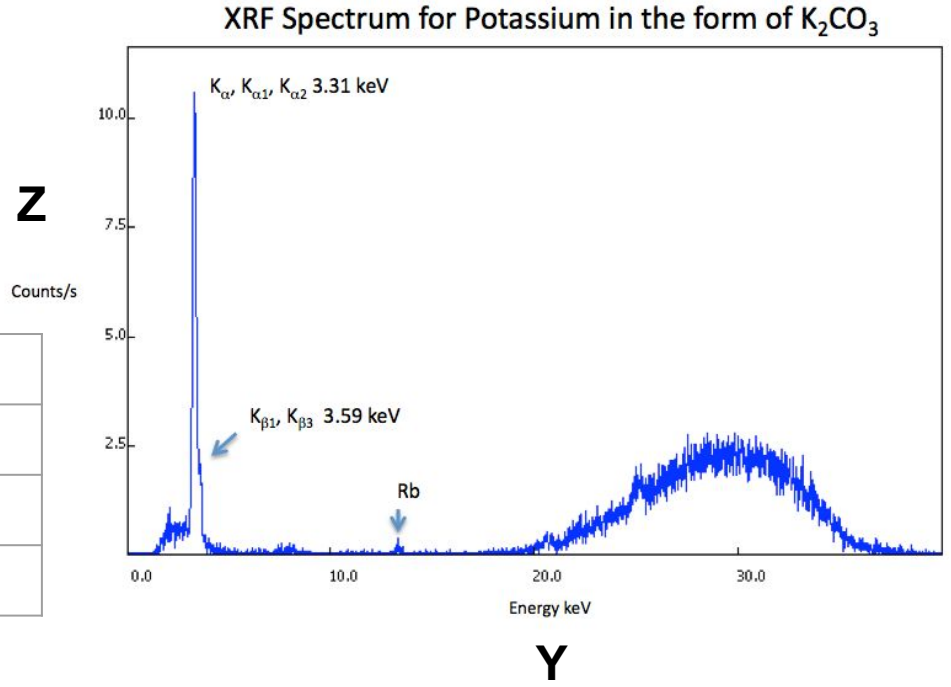
- pgpointcloud
  - *Four* different types of compression
    - zlib
    - sigbits
    - rle
    - laz
  - `Points` (X, Y, Z) are combined into `Patch`es (`Point[]`)



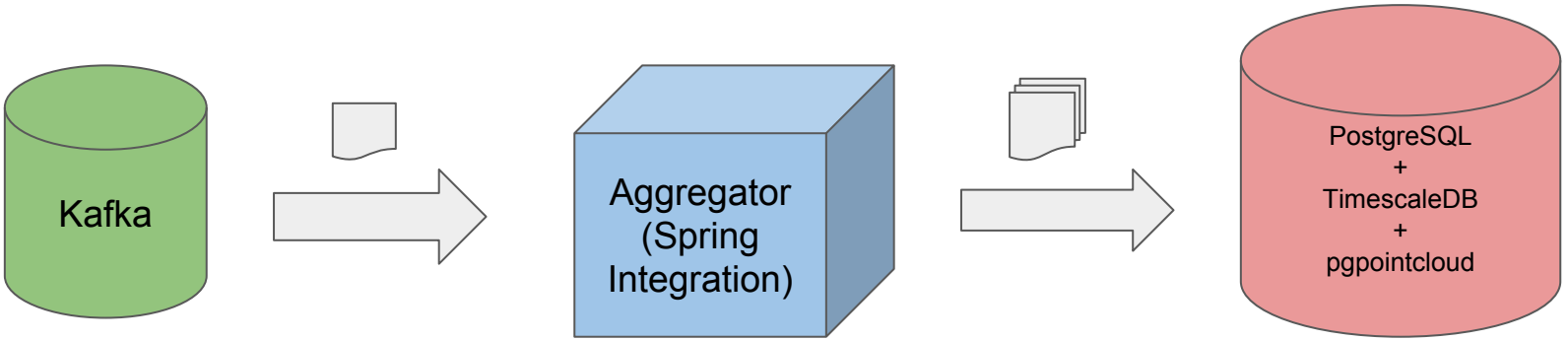
How we tried to solve it

- pgpointcloud
  - X: `uint64_t` (time in millis)
  - Y: `uint16_t` (bin/channel)
  - Z: `uint16_t` (count)

X (time)	Y (bin/channel)	Z (count)
1553136615	2	10
1553136615	13	1
1553136615	30	3



How we tried to solve it



Results

```
create table Spectrum (
  sensorId UUID NOT NULL,
  starttime TIMESTAMP WITH TIME ZONE,
  endtime TIMESTAMP WITH TIME ZONE,
  spectrum pcpatch(3),
  PRIMARY KEY(sensorId, starttime)
);
```

- ~80% compression!
  - Compressing across three dimensions: time, channel, and energy
- 60x reduction of rows
- Stable insertion rate
- Full visibility of the data via SQL
  - `PC_Explode(p pcpatch)`
  - `PC_FilterGreaterThan(p pcpatch, dimname text, float8 value)`

Other

- Can use exclusion constraints if you don’t want overlapping patches
- Try Citus with TimescaleDB
- pgpointcloud / TimescaleDB are orthogonal
- Experiment with aggregating different time ranges
- Experiment with aggregating between different sensors

Conclusion

- Know your data
- Do not fear PostgreSQL extensions
- Greater than the sum of their parts
- Dockerfile available at https://github.com/twosixlabs/docker-postgres-pointcloud

The End

- Questions?
- Thank you!
- We are hiring!
  - https://twosixlabs.com/careers