Data Architecture for Sensor Network

Jan Jezek
Faculty of Applied Sciences, Department of Mathematics
University of West Bohemia in Pilsen
Univerzitní 22, Plzeň 306 14, Czech Republic
`jezekjan@kma.zcu.cz`

Keywords: sensor network, data architecture, NoSQL, PostgreSQL, PostGIS, Apache Cassandra

Abstract

Fast development of hardware in recent years leads to the high availability of simple sensing devices at minimal cost. As a consequence, there is many of sensor networks nowadays. These networks can continuously produce a large amount of observed data including the location of measurement. Optimal data architecture for such propose is a challenging issue due to its large scale and spatio-temporal nature.

The aim of this paper is to describe data architecture that was used in a particular solution for storage of sensor data. This solution is based on relation data model – concretely PostgreSQL and PostGIS. We will mention out experience from real world projects focused on car monitoring and project targeted on agriculture sensor networks. We will also shortly demonstrate the possibilities of client side API and the potential of other open source libraries that can be used for cartographic visualization (e.g. GeoServer). The main objective is to describe the strength and weakness of usage of relation database system for such propose and to introduce also alternative approaches based on NoSQL concept.

Application concept and requirements

The aim of this section is to describe software solution focused on sensor data storage and visualisation. The application has these goals:

- System provides storage of sensor data
- System is general for various kinds of sensors including moving sensors
- System provides a simple web API for inserting observations, positions and alerts and also for data acquisition.
- System id scalability, highly available, fault tolerant

In the scope of this article, we will describe how we deal with these requirements. The overall concept is shown on fig 1. When data are measured by particular sensor, they are transmitted into the mobile unit (see for a better explanation of the hardware) that transmits the data using GPRS. On the main server, there is a web service interface that enables insertion of data to the main database. The data from the database are then visualized by various ways.

It is worth to mention that the solution based on central data warehouse is not the only possible approach. The alternative might be to store the data on particular mobile unit (as described in [4] or [5]) and query the sensor network to transmit just the data that we are interested in. We choose to transmit all the data immediately and store it outside the sensor network for this reasons:

- Sensor are as simple as possible and doesn't provide much pressing or memory capacity. The aim is to keep the senors and Mobile units as simple and as cheap as possible.
- Mobile unit can store the data, but it is very difficult to ensure high availability of such unit. Mobile units are located in the field and failure from miscellaneous reason might occur in any time.
- The Mobile units are connected to Internet using GPRS (3G). It might happen that unit can lost the signal on some places.
- The system uses different sensors and units from different providers, and it is hard to manage any more complicated software on them. The units might be located in cars of customers for example, and any upgrade will require to contact them.

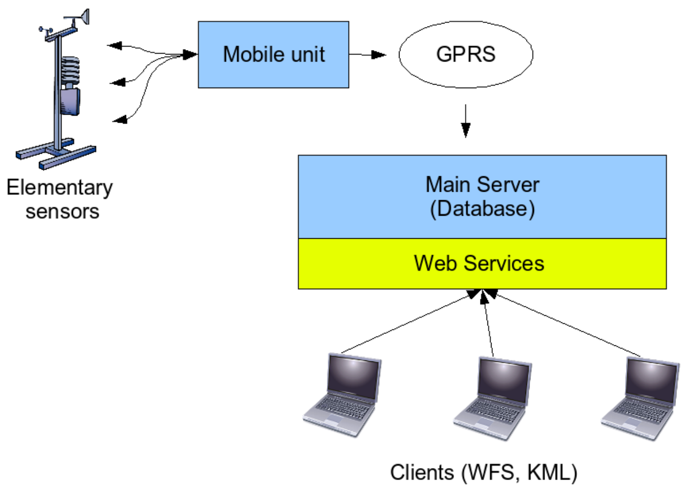


Figure 1. Basic concept

Database solution

PostgreSQL + PostGIS

The effective storage for sensor data is essential. Nowadays we do have about 40 units but the network should be much larger in the future. Such network provides this amount of data:

- Nowadays about 40 – 60 units
- Each unit has about 5 – 10 sensors
- Each unit is sending observation approximately each 15 seconds
- Each unit sends position each 15 seconds

The result is that we have about two millions of observations in one day and about 300 000 of positions in one day.

The database has to provide fast access especially to these data:

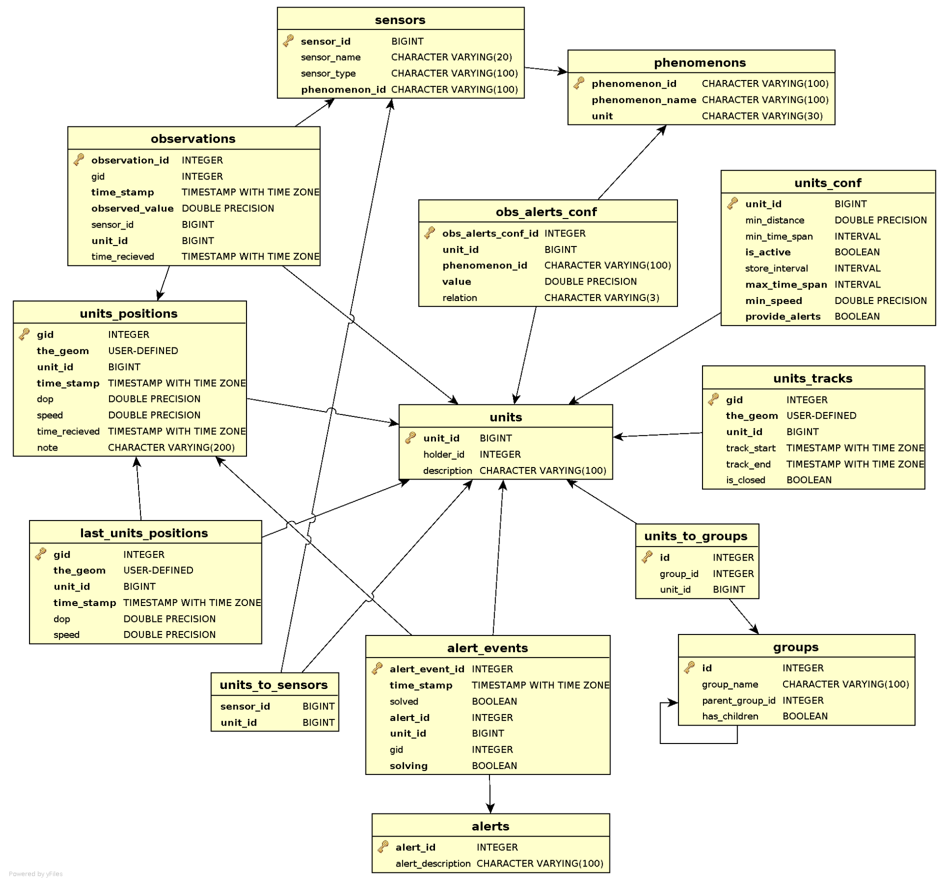


Figure 2. Database schema

- Last positions of unit
- Units tracks (Array of time sorted positions limited by a particular event like status of engine or similar)
- Average values of observations

As the data are growing really fast, the performance of 'Join' based query is low and does not scale well. For that propose we use denormalization and we are generating query results in time of insertion of position or measurement rather than in time of query. For that propose we have implemented many triggers and rules. The drawback of such approach is data duplicity and hardly manageable consistency.

The database schema (fig. 2) contains these main tables:

- *units* – List of Mobile Units located in sensor network.
- *sensors* – List of sensor types located in sensor network, sensors are attached to units (m:n)
- *phenomenons* – Phenomenons of measured values
- *observations* – particular sensor observations

- *alert_events* – Real events of alert situation (e.g. temperature is exceeding predefined threshold)
- *alerts* – General description of alert situation that should be monitored

NoSQL – Appache Cassandra

As has been described – to achieve proper query performance when using PostgreSQL and PostGIS we finally have to denormalise the data and maintain data in the shape that follows the query rather then data relations. For example we have to maintain table that contains tracks of each unit (By track we mean LineString of time sorted positions, where the track start and ends in a particular time according to the value provided by a sensor that is attached to engine ignition for example). Such a table is updated when the position is inserted. When we change the geometry of the track we should also change the geometry of the particular position stored in units_position table what makes the design unclean and what makes the maintenance harder.

In recent years there has been new concept proposed for such issue that is generally called NoSQL. For specific reason we started to implement a prototype using Apache Cassandra database system. One of key feature of this DBMS is that the data design is focused on queries from the scratch and the denormalization is a key concept. It dos not solve the problem with data duplicity, but as a trade of it offers much easier partitioning and high availability. Another reasonable fact is that this DBMS is extremely scalable and fault tolerant.

A drawback of Apache Cassandra is the absence of spatial data support. Good database design and effective spatial indexing is under research nowadays and will be describe in the near future.

Clients

Data acquisition through HTTP GET API

Main interface for client side application is designed by a set of web services that acts as middleware between client application and database. These services provides:

- Metadata about sensor network -There are important information about topology of sensor network, positions of particular sensors, types of sensors and observed phenomenons, accuracy and frequency of measurement.
- Observed values – From metadata users can choose what set of values are required. By using proper service observed values can be requested. Such request can be parametrized by time period, location, phenomenon etc. There is also possible to define the output format (e.g. xml, json, csv or chart raster file)

These web services has been implemented in Java as a server side application. Services provides also efficient mechanisms to achieve good performance. Frequently requested data are pregenerated in the database or cached after they are firstly requested. Service also provides basic calculations like providing average values for time intervals (minutes, hours, days etc).

The database and server side application also provides the capability for users authentication and authorization. Unites and system users can be assigned to hierarchy of groups and in this way we can control what data can be provided to whom.

From a technical point of view these services are implemented to be as simple as possible so that they can be easily used in client side applications. Services are provided by HTTP GET and POST by sending set of proper Key value pair values.

Sample of metadata request:
http://test.sensors.lesprojekt.cz/DBService/DataService?Operation=GetUnits

Sample part of response (just for one Mobile Unit):

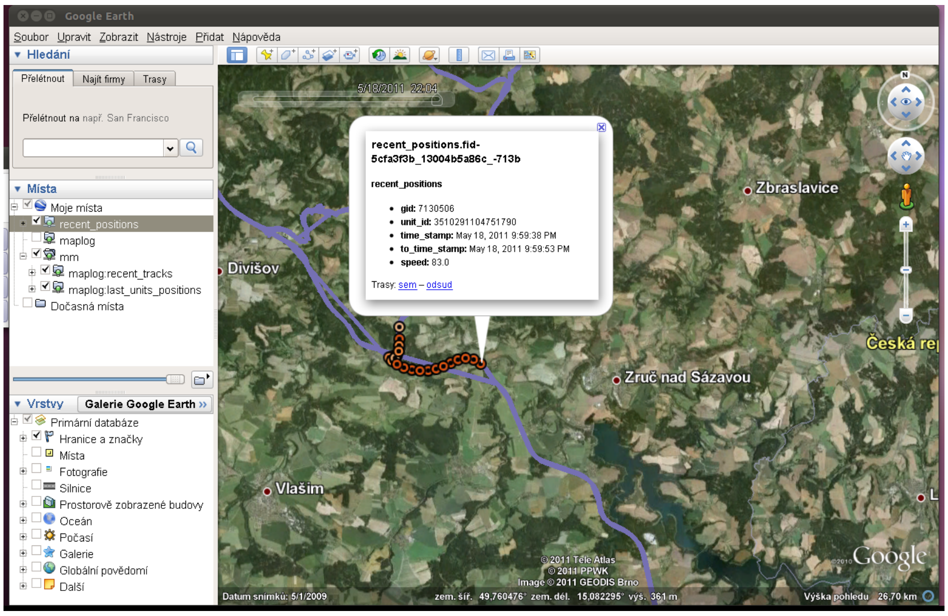
```
[ { "lastpos" : { "position" : { "gid" : 127096,
            "group_id" : 1,
            "speed" : 0,
            "time_stamp" : "2010-06-23 18:34:54+02",
            "unit_id" : 101502765075,
            "lon" : 14.1536,
            "lat" : 50.8486,
            "srid": 4326
          }
      },
    "sensors" : [ { "firstObservationTime" : "unknown",
          "lastObservationTime" : "unknown",
          "phenomenon" : { "phenomenonName" : "Rssi",
              "unit" : "dB"
            },
          "sensorId" : 380011502765075,
          "sensorName" : "Rssi 200",
          "sensorType" : "signal strength\n"
        },
        { "firstObservationTime" : "2010-07-29 10:29:40+02",
          "lastObservationTime" : "2011-04-06 14:27:53+02",
          "phenomenon" : { "phenomenonName" : "retransmission count",
              "unit" : "-"
            },
          "sensorId" : 360041502765075,
          "sensorName" : "Vlit retrans 200",
          "sensorType" : ""
        }],
    "unit" : { "description" : "Mobile Unit v2",
        "holderId" : 0,
        "unitId" : 101502765075
      }
  } ]
```

This interface can be understood as simplified variant of OGC specification -*Sensor Observation Service*. The reason why we decided to implement our own standard instead of using the OGC specification is mostly because of significant complexity of that standard.

Other clients

As has been already mentioned, the main storage of data is PostgreSQL with PostGIS. This storage supports the OpenGIS Simple Feature Specification for SQL, so it can be quit easily contend to any desktop (e.g. QGIS, uDig) or server side client (GeoServer, Mapserver). As long as the desktop clients does not provide capabilities to handle effectively large dataset, it is much better to utilise server application as middleware so that they provide caching and generalisation.

One of example of utilising this interface is GoogleEarth client (fig. 3). One of nice feature of GoogleEarth is the ability to deal with temporal data by using time slider.



recent_positions.fid-5cfa3f3b_13004b5a86c_-713b

recent_positions

- **gid:** 7130506
- **unit_id:** 3510291104751790
- **time_stamp:** May 18, 2011 9:59:38 PM
- **to_time_stamp:** May 18, 2011 9:59:53 PM
- **speed:** 83.0

Trasy: sem – odsud

Figure 3. Google Earth visualisation

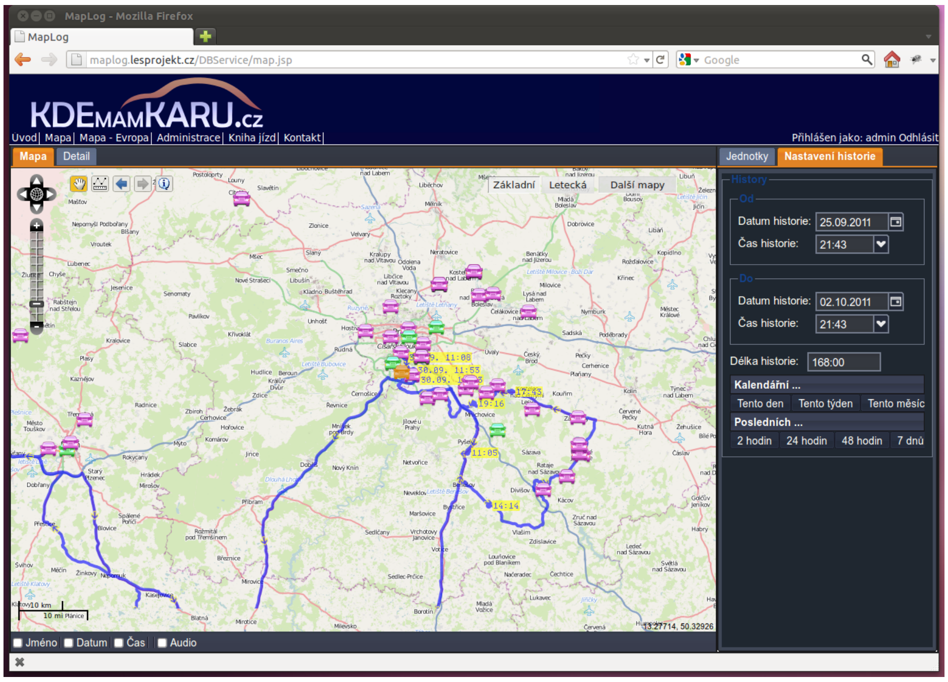
For the particular project (focused on car monitoring) there has been client implemented on the top of HSLayers library (javascript library focused on web mapping). The example is shown in Fig. 4 (just in Czech version).

Conclusion

Paper briefly summarises the basic concept of sensor base application. This application has been developed and is used in several real world project focused on moving object monitoring and meteorological analyses for agriculture proposes. We tried to mention the pros and cons of used concept and discus the proposed solution. A key part of ongoing research is the enhancement of scalability by migration to NoSQL concept. This is actually under progress a has not been finished yet.

References

1. Brewer, E.: Principles of Distributed Computing (PODC 2000): "Towards robust distributed systems." Portland, OR., July 2000.USA. ISBN 978-0-7695-3858-7 (2009)



Datum historie: 25.09.2011
Čas historie: 21:43

Datum historie: 02.10.2011
Čas historie: 21:43

Délka historie: 168:00

Figure 4. HSLayers based client

2. Eben, H.: Cassandra The definitive Guide. O'Reilly, 2010, ISBN 978-1-449-19041-9

3. Lakshman, A.; Malik, P.: Cassandra — A Decentralized Structured Storage System. Cornell University. Retrieved 13 November 2009.

4. Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong: "TinyDB: an acquisitional query processing system for sensor networks", ACM Trans. Database Syst., Vol. 30, No. 1. (March 2005), pp. 122-173.

5. Philippe Bonnet, J. E. Gehrke, and Praveen Seshadri: Querying the Physical World. IEEE Personal Communications, Vol. 7, No. 5, October 2000, pages 10-15. Special Issue on Smart Spaces and Environments.