Two-Tier Architecture for Web Mapping with NoSQL Database CouchDB

Mario MILER, Damir MEDAK and Drazen ODOBASIC

Abstract

Two of the greatest challenges of mobile data collecting applications are replication and synchronicity. This paper deals with these challenges and offers a new solution: two-tier architecture using CouchDB. CouchDB is a RESTful non-relational (NoSQL) document-oriented database with the ability to serve web pages directly from the database. CouchDB is acting as a web server and data storage on the server side and as a web browser on a client side. Although cloud functions on a mobile device can be used, the main advantage of CouchDB is the ability to work offline with both data and application, replicating the results back to the server. Combining such a solution with GPS we get simple mobile GIS data collector in a standard browser application. Many smartphones and mobile devices can be transformed to mobile data collectors with ease. In this paper, we will show the fundamental CouchDB concepts as a data and application storage.

1. Introduction

Web mapping is the easiest and the most widely accessible usage of GIS technology. During the last few years, we witnessed the "Google Map phenomenon," which opened a new market demand for the integration of mainstream GIS (Andrews, 2007). As the GIS industry tries to supply this demand, new technologies arise rapidly migrating to the mobile market. The development of new mobile devices with Internet connections provides new opportunities for web mapping. There are two commonly used approaches in mobile data collection: through standalone application and a standard web browser. Most of these solutions are based on database-server-browser/application architecture (later referred as three-tier architecture), and can operate both in online and offline mode. In online mode, this model requires stable high-bandwidth Internet connection for retrieving data from a database. Since mobile devices cannot guarantee such connection will always be available, offline solutions have to be implemented. Offline solutions require synchronization of data when a connection is established. Synchronicity and replication is the process of sharing information between multiple redundant storage sources. When using this approach with a standard Internet browser, an offline solution is not possible because it requires a constant connection to the server. Few commercial solutions available provide applications for offline GIS data viewing and editing. These solutions are often very expensive and often do not follow standard communication protocols needed for data interoperability. This paper proposes a data model and alternative architecture for mobile/desktop web mapping solution. The prototype application is based on open source NoSQL database CouchDB and a web application written in standard HTML and JavaScript programming language.

This paper is structured as follows: the next section will present basic concept of prototype application. Section 3 describes used technologies and protocols. Section 4 describes the structure and retrieval methods for raster and vector data in the database. In section 5, we will assess and discuss our solution. The conclusion is provided in section 6.

2. Objectives

The main research objective was to create an application that would store, provide and replicate spatial data using as little technology as possible. The solution has to be simple enough to use and develop to implement standard, open communication protocols, and not be bound to one operating system. This requires basic architecture, which is independent of constant Internet connection and which is not bound to a single operating system. Such system should be able to work offline and replicate the data when connection is established. Open and standard communication protocols must be used.

The easiest way to accomplish this is to use a standard web browser, which supports JavaScript. Such web browsers are common on all operating systems. Usually web mapping is achieved with standard web mapping services like WMS, WFS, or TMS, but they require a constant Internet connection. In areas where such connection is weak or not available, this approach is not possible. The solution is to use local data storage serving data and to replicate it once the connection is available. We propose a NoSQL implementation leaning on CouchDB database with its ability to serve web pages directly from the database and data via REST interface. Geospatial data are integrated within the same application.

3. Applications and methods

3.1. The NoSQL movement

NoSQL is a term in information technologies that describe database management systems, which depart from classic RDBMS (Relational Database Management Systems) altogether, or in some parts. The term was first used for lightweight open source databases that did not use SQL (Structured Query Language) as a database language. The name “NoSQL” could indicate that these databases do not support SQL, but in this case it actually means “Not Only SQL”. In his 2008 article, Strozzi claimed that because the NoSQL movement departs from the relational model altogether, it would more appropriately be called 'NoREL'. The term was re-introduced as a general term in 2009 by Eric Evan at Rackspace. In academic research papers, these databases are referred to as a structured storage or non-relational database. As opposed to SQL as a standard querying language, most of the NoSQL databases implement MapReduce algorithm for querying and extracting relevant data.

MapReduce is a programming model that enables the easy development of scalable parallel applications to process vast amounts of data on large clusters of commodity machines (Yang at al., 2007). The model implements only two functions commonly used in functional programming: *map* and *reduce*. It was introduced in a paper by Google Inc. (Dean, Ghemawat, 2004) to support distributed computing on large data sets on clusters of computers. The combination of the map and reduce functions in CouchDB is called a *view*. Map function is called once for every document in the database, deciding whether to skip or *emit* key/value par. CouchDB views are stored in rows and are indexed by the emitted key, which makes data retrieving very fast, even with millions of rows. When writing a view, the goal is to adapt future searches from the key and data searched to the emitted value of the key/value pair.

NoSQL is often considered a synonym for open source databases, but NoSQL is not about open source. The business model does not matter here; it is the technology (Kellogg, 2010).

Traditional RDBMS databases like PostgreSQL, Oracle, SQL Server or MySQL rely on ACID (Atomicity, Consistency, Isolation and Durability) properties, which NoSQL databases lack or partially lack. NoSQL databases are developed to run on a cluster of "cheap" and commodity servers and personal computers. In order to achieve this some NoSQL databases neglect some of the ACID properties. ACID properties do not contradict or negate the concept of NoSQL, but there seems to be a trend following that opinion. NoSQL does not mean that ACID properties are not implemented (Orend, 2010).

NoSQL databases are fundamentally schema-free key-value pair datastore. It is a direct alternative to traditional RDBMSs, which store data in terms of tables. All data are stored in a table, and when modelling complex data structures, an individual piece of data may be split across one or more tables. For some applications and data types, this is a perfectly logical and reasonable way of approaching and storing your data. For some applications the table structure does not map very well to the data you want to store (Couchone, 2010). NoSQL is a part of a broader trend in database systems - specialization. Traditional RDBMSs work very well for most applications but are not the best solution for everything (Varley, 2009).

There are several classifications of NoSQL databases, but all current implementations belong to four major classes (Varley, 2009):

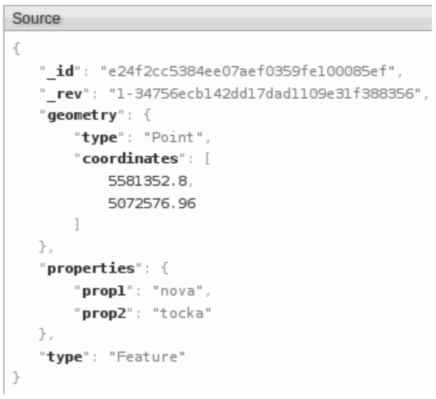
- key-value stores, also known as distributed hash tables, e.g. *Amazon's Dynamo, Voldemort, Berkeley DB, Hadoop*
- bigtable, also known as multi-dimensional tabular systems, e.g. *Google's Bigtable, Hypertable, HBase*
- document-oriented databases, e.g. *CouchDB, MongoDB*
- graph-oriented databases, e.g. *Neo4j, AllegroGraph.*

3.2. CouchDB

CouchDB is a part of the Apache Software Foundation, which provides organizational, legal, and financial support for a broad range of open source software projects. CouchDB is developed in Erlang, which is a robust functional programming language designed at the Ericsson Computer Science Laboratory for building concurrent, distributed systems. Erlang is a programming language designed. Open-source Erlang is being released to help encourage the spread of Erlang outside Ericsson (Armstrong, Däcker, Lindgren, & Millroth, 2010). Initially, CouchDB was developed in C++, but from 2006, development was moved to Erlang because of its concurrency control, fault tolerance, and distributed applications which suited perfectly for CouchDB usage.

CouchDB is a multi-platform schema-free document-oriented database with an optimistic replication mechanism (Anderson, Lehnardt, & Slater, 2009). Each document is uniquely named in the database with its ID. The main segment of data in CouchDB is a single document. In traditional RDBMSs, it could be seen as one row of data in one table. A document can consist of any number of fields and attachments, which can differ from document to document for the same database. Document fields must be uniquely named, and data are not limited by size or number of elements. Each document can have a different number of fields or no fields at all (with the exception of a unique id). Basically, this is what makes CouchDB a schema-free database. In traditional RDBMSs, the schema is mostly fixed and highly structured. Very often it is hard to change or update schema after data has been entered into the tables. In CouchDB, (NoSQL databases in generally) there is no such constraint. This makes NoSQL databases ideal for systems where the hard structure of data is changing. CouchDB is designed to store large amounts of semi-structured, document- oriented data.

Documents in CouchDB are stored in a single structure, JSON (JavaScript Object Notation) format. The JSON format allows for a complex structure of fields, arrays, objects, and scalar types, which can be combined into an entire record (Orend, 2010). JSON is a text format for the serialization of structured data. It is derived from the object literals of JavaScript, as defined in the ECMAScript Programming Language Standard, Third Edition (Crockford, 2006). ECMAScript is the scripting language standardized by Ecma International in the ECMA-262 specification and ISO/IEC 16262. A few years ago, the geospatial community has suggested a geographic dialect of JSON, called GeoJSON. The idea was to standardize the way of distributing spatial data over the web. GeoJSON can be used to represent geometry, feature, a collection of geometries, or collection of features. Each feature (geometry object) can contain additional properties. Supported geometry types in GeoJSON are Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and Box. A complete GeoJSON data structure is always an object (in JSON terms) (Schaub, Doyle, & Daly, 2008).



```
{
    "_id": "e24f2cc5384ee07aef0359fe100085ef",
    "_rev": "1-34756ecb142dd17dad1109e31f388356",
    "geometry": {
        "type": "Point",
        "coordinates": [
            5581352.8,
            5072576.96
        ]
    },
    "properties": {
        "prop1": "nova",
        "prop2": "tocka"
    },
    "type": "Feature"
}
```

Figure 1 Example of point in GeoJSON data structure stored in CouchDB

3.3. CouchApp

A *CouchApp* is a JavaScript and HTML5 application that can be served directly to the browser from CouchDB database, with no other applications in the middle. A standard way of writing a dynamic database-backed web application is by using the three-tier architecture consisting of the client, application server and database.

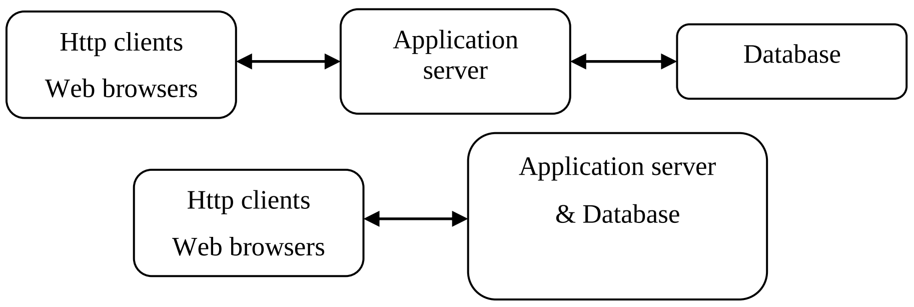


Figure 2 Three-tier and two-tier architecture

Three-tier architecture (Ramirez, 2000):

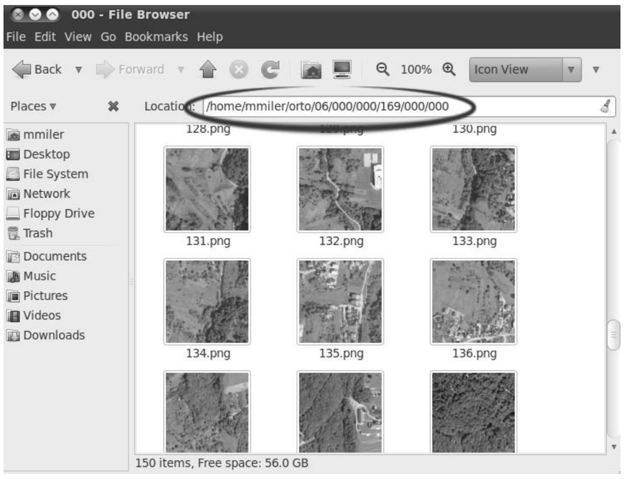
- Tier 1: the client contains the presentation logic, e.g. Internet Explorer, Mozilla Firefox, Google Chrome etc. This application is also known as a thin client.
- Tier 2: the middle tier is also known as the application server, which provides the processes logic and the data access.
- Tier 3: the data server provides the data

Most of web applications written today are based on three-tier architecture. *CouchApp* is a set of scripts that allows complete, stand-alone CouchDB applications to be built using just HTML and JavaScript (Anderson, Lehnardt, & Slater, 2009). *CouchApp* enables web applications to be served directly from the database as they would be served through any other web server. This is mostly possible because CouchDB has a RESTful interface. It means that one can communicate with CouchDB as they would with any webserver. The only difference that response is not a standard HTML webpage as it might be expected from a standard webserver but data (documents) in JSON format. The fact is that CouchDB has a protocol that is universal, and not binary like the one of common databases such as MS SQL Server, Oracle or MySQL. Although CouchDB has a RESTful API for accessing the database, the entire API is not RESTful (Anderson et al., 2009). Essentially, database resources can be accessed via ordinary web browser (e.g. Mozilla Firefox, Internet Explorer, Chrome etc.) without any service in between. The term *Representational State Transfer* was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. “REST enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate semantics and exchange information, and responses explicitly indicate cacheability.” (Fielding, 2000).

3.4. Openlayers and TileCache

Openlayers is an open source geospatial AJAX toolkit designed to display and manage spatial data in a web browser, originally developed by MetaCarta. It is entirely written in JavaScript and has no requirement from the client side except the web browser, which supports JavaScript. OpenLayers implements industry-standard methods for geographic data access, such as the OpenGIS Consortium's Web Mapping Service (WMS) and Web Feature Service (WFS) protocols, GeoJSON, KML, GML and other formats. It is agnostic of server-side technologies, which means the technology used on the server-side is insignificant as long as it follows the standard communication protocols.

TileCache is an implementation of a WMS-C compliant server, which was also developed by MetaCarta. It provides a Python-based WMS-C/TMS server, with pluggable caching mechanisms and rendering backends like MapServer or Mapnik. Its primary function is to speed up the WMS request by caching generated tiles on disk and can speed up access to WMS server by factors of 10-100, or more (TileCache, 2010). Tiles are stored on disk in a structured directory hierarchy, e.g. *layer/0/000/001/053/000/021/052.png*.



Location: /home/mmiler/orto/06/000/000/169/000/000

150 items, Free space: 56.0 GB

Figure 3 Example of stored tiles on disk

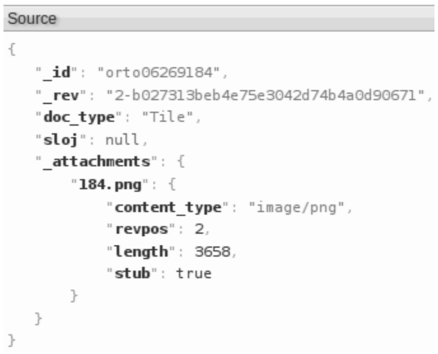
4. Prototype application

The software component used to create the prototype application is CouchDB 1.0.1, Openlayers JavaScript mapping library, Mapnik library for generating tiles, TileCache for storing tile structure, and Python programming language for scripting. As one of the objectives was to create an application that was simple to develop, only basic knowledge programming in JavaScript and Python was necessary for all the coding.

4.1. Raster data (Map tiles)

For mapping applications, it is important to have access to both raster and vector data. Displaying georeferenced raster data in a web browser is difficult if these rasters are not served with some web mapping service. WMS is commonly used, but it is not appropriate in this case because it requires a constant, active service on the device or connection to the server. In order to use large raster images efficiently in a web browser, raster images need to be tiled in ‘mosaic-like pieces’. Storing and reading large raster images as a whole would drastically slow down the application, especially for a mobile device. Mapnik and TileCache were used for tiling and rendering raster data. Mapnik is an open source mapping library for map rendering raster and vector data used in WMS, TMS or other web mapping services. TileCache was used for caching those rendered tiles from Mapnik to structured directory hierarchy on a disk. Later, this structured hierarchy will be very important for retrieving a particular tile.

Using the Python programming language and *Couchdbkit* library, all rendered tiles were imported into CouchDB as a document attachment with a unique ID. This unique ID is important for retrieving a particular tile from the database. The unique ID is a string concatenated of layer, zoom level, structured TileCache tile identification and name of the tile. For example, tile on a disk with the directory location *layer/0/000/001/053/000/021/052.png* would correspond to *layer0000001053000021052*. This string uniquely identifies one tile in a particular layer. For the sake of simplicity, the prototype application uses a short and unique ID, which does not include redundant zeros (see Figure 4).



Source

```
{
    "_id": "orto06269184",
    "_rev": "2-b027313beb4e75e3042d74b4a0d90671",
    "doc_type": "Tile",
    "sloj": null,
    "_attachments": {
        "184.png": {
            "content_type": "image/png",
            "revpos": 2,
            "length": 3658,
            "stub": true
        }
    }
}
```

Figure 4 One tile stored as an attachment in CouchDB document

With all tiles imported into the CouchDB database, we can use CouchDB’s ability to access every tile via standard REST interface directly from the CouchDB database using previously generated unique ID, e.g. http://localhost:5984/my_app/layer0000001053000021052/052.png

An Openlayers plugin was necessary to use restfulness of CouchDB with Openlayers mapping library. The plugin is reading the tiles in the same way it would read the tiles from any other web mapping service. It was possible to calculate tile ID based on the current view in application, because TileCache creates a structured directory hierarchy based on tile geolocation, using that same algorithm.

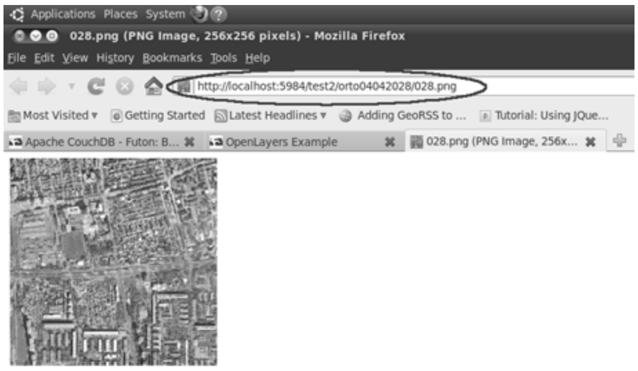
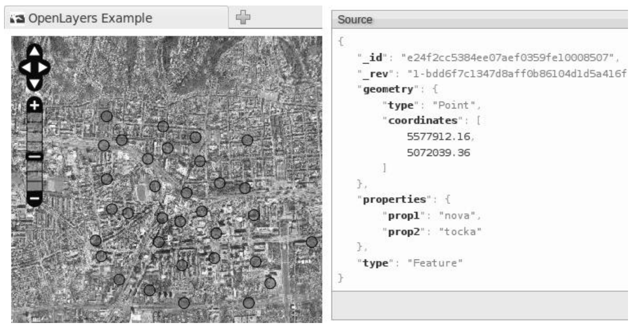

Applications Places System
028.png (PNG Image, 256x256 pixels) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:5984/test2/orto04042028/028.png
Most Visited ▾ Getting Started Latest Headlines ▾ Adding GeoRSS to ... Tutorial: Using JQue...
Apache CouchDB - Futon: B...
OpenLayers Example
028.png (PNG Image, 256x...

Figure 5 Accessing one tile via REST interface

4.2. Vector data (JSON)

Vector data are a collection of points, lines and polygons. CouchDB could have been used for storing standard GIS data formats as an attachment (similar to raster data). Such solution would have problems with reading and writing to the database. The GeoJSON format was used for storing of all vector data because CouchDB is basically the JSON storage. Slight modification had to be made using JavaScript to format CouchDB JSON document to fit GeoJSON standard used in Openlayers library. The current prototype application is capable of rendering of saved vector data and input new data depending on location. It is straightforward to develop editing and deleting of data.

Visualization of georeferenced vector data directly from the database is simpler than raster data. The problem occurs if a large number of objects are to be displayed in a web browser. Web browsers are not optimized for displaying of vector data.



```
{
   "_id": "e24f2cc5384ee07aef0359fe10008507",
   "_rev": "1-bdd6f7c1347d8aff0b86104d1d5a416f
   "geometry": {
      "type": "Point",
      "coordinates": [
         5577912.16,
         5072039.36
      ]
   },
   "properties": {
      "prop1": "nova",
      "prop2": "tocka"
   },
   "type": "Feature"
}
```

Figure 6 Dots show vector point data from local CouchDB storage (left) and one point stored in JSON format (right)

Using Geolocation API that is implemented in all modern web browsers, it is possible to store current location into the database as a point or a waypoint data. The Geolocation API defines a high-level interface (usually web browser) to the location information (longitude and latitude) associated only with the device hosting the implementation. The API itself is agnostic of the underlying location information sources. Common sources of location information include Global Positioning System (GPS) and location inferred from network signals such as IP address, RFID, WiFi and Bluetooth, MAC addresses, and GSM/CDMA cell IDs, as well as user input. There is no guarantee that the API will return the device's actual location (W3C, 2010).

5. Discussion

During the development of this prototype application, we have realized that although it uses basic technology, it would take a lot more work to create a full-working application. Even some basic features that are standard in other mobile data collection applications like exporting and uploading new data would require a lot of customization programming.

At any time, application should not display a large number of vector data. Web browsers, especially older versions, are not optimized for displaying large quantities of vector data. Mobile devices do not have processors fast enough for displaying large amount of vector data without significant performance loss. An alternative approach would be to render a part of vector data to the raster format. The problem is to develop a service that would intelligently render only required raster tiles from vector data that are not needed to be in vector format.

The largest disadvantage of rasterized and rendered data as tiles is the required amount of disk storage space. Although current mobile devices have more disk space than before, it is still not enough, especially if there are more layers. This problem can be resolved by uploading the area of interest only, which requires additional customization at both the server and client side.

The proposed solution, in its current implementation, is designed for lightweight applications. Major development effort is needed for the preparation of an enterprise solution. Nevertheless, CouchDB has proved to be an interesting solution for storing and providing spatial information without any service between the database and clients.

6. Conclusions

The NoSQL movement and CouchDB in particularly are changing the perception of databases and the interaction between clients and databases. The prototype application is demonstrating the ability of a web browser and CouchDB to serve as a data collecting application. The use of standard GIS functions could be implemented by adding other JavaScript libraries, which could be the next step of the development. Using CouchDB, which is available on almost all operating systems, this application is fully platform-independent. Future will bring more NoSQL databases to the mainstream GIS because of its versatility, performance and openness.

7. References

Anderson, J. C., Lehnardt, J., & Slater, N. (2009). *CouchDB: The Definitive Guide.* Sebastopol: O'Reilly.

Andrews, C. (2007). *Emerging Technology: AJAX and GeoJSON.* Retrieved December 14, 2010, from Direction Magazine: http://www.directionsmag.com/authors/christopher-j-andrews/122075

Armstrong, J., Däcker, B., Lindgren, T., & Millroth, H. (2010). Retrieved January 27, 2011, from Erlang.org: http://www.erlang.org/white_paper.html

Couchone. (2010). *How to Move from MySQL to CouchDB.* Retrieved December 9, 2010, from http://blog.couchone.com/post/2145537100/how-to-move-from-mysql-to-couchdb-part-1

Crockford, D. (2006). *The application/json Media Type for JavaScript Object Notation (JSON).* Retrieved December 9, 2010, from http://tools.ietf.org/html/rfc4627

Dean, J., & Ghemawat, S. (2004). *MapReduce: Simplified Data Processing on Large Clusters.* San Francisco.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures (Doctoral dissertation).* Irvine: University of California.

Kellogg, D. (2010). *Six Thoughts on The NoSQL Movement.* Retrieved December 8, 2010, from http://kellblog.com/2010/06/18/six-thoughts-on-the-nosql-movement/

Orend, K. (2010). *Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace an Object-relational Persistence Layer.* Munich: Technical University Munich, Faculty of Informatics.

Ramirez, A. O. (2000). *Three-Tier Architecture.* Retrieved December 2010, from Linux Journal: http://www.linuxjournal.com/article/3508

Schaub, T., Doyle, A., & Daly, M. (2008). *GeoJSON Specification draft version 6.* Retrieved December 14, 2010, from http://wiki.geojson.org/GeoJSON_draft_version_6

Strozzi, C. (2008). *NoSQL A Relational Database Management System.* Retrieved December 2010, from http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home Page

TileCache. (2010). *TileCache -- Web Map Tile Caching*. Retrieved December 2010, from http://tilecache.org/

Varley, I. T. (2009). *No Relation: The Mixed Blessings of Non-Relational Databases (Master's Thesis).* Austin: The University of Texas.

W3C. (2010). *Geolocation API Specification.* Retrieved December 2010, from W3C Web site: http://dev.w3.org/geo/api/spec-source.html

Wikipedia. (2010). *NoSQL*. Retrieved December 2010, from Wikipedia, The Free Encyclopedia: http://en.wikipedia.org/wiki/NoSQL

Yang, H.-C., Dasdan, A., Hsiao, R.-L., & Parker, D. S. (2007). Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters. *SIGMOD.* Beijing.