FOSDEM '13

Perl & NoSQL Focus on MongoDB

Jean-Marie Gouarné
http://jean.marie.gouarne.online.fr
jmgdoc@cpan.org

AGENDA

- A *NoIntroduction* to *NoSQL*
- The “document store” data model
- MongoDB at a glance
- MongoDB Perl API basics
- A few more advanced queries
- Server side code execution

Introduction

- For a real NoSQL focus, wait for *https://fosdem.org/2013/schedule/track/nosql*
- NoSQL is a no-concept ; it's essentially an advocacy group ; it's certainly not a technology
- The NoSQL catalogue and taxonomy are out of my present scope ; today I focus on the “document stores” category, illustrated by MongoDB, and its Perl connection
- NoSQL is cool and mature ; unfortunately it's essentially a NoPerl world, for non-technical reasons

A few common characteristics

- Non tabular logical schemas
- No standard data manipulation language (for the time being)
- Flexible data structures
- “Eventual consistency” (no ACID mechanism)
- Unlimited scalability (hopefully)
- Native replication/distribution mechanisms
- Open source (with commercial support)
- …

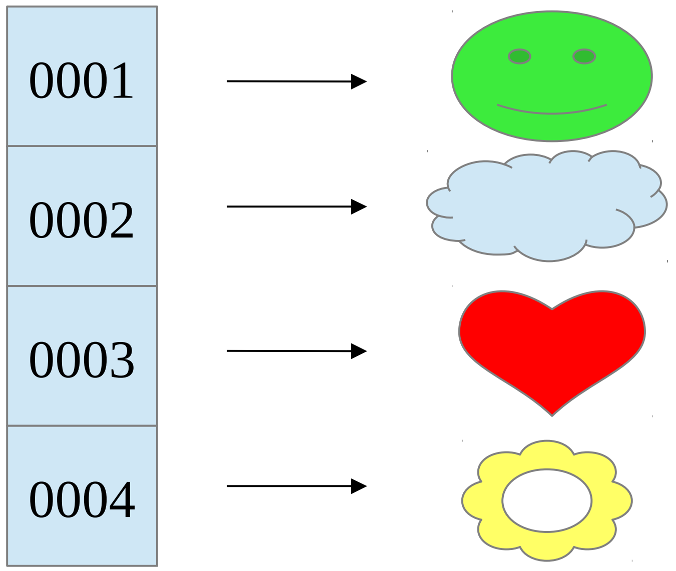
Not intended to replace SQL (...for now)

Essential messages

- data may be persistently stored without a previously defined, tabular schema ; heterogeneous logical structures are allowed in the same data set ; NoSQL should mean NoDDL
- a logical data record may be much more sophisticated than a “table row”, but multi-objects joins, when needed, are supported by the application, not by the data engine
- the “commit/rollback” transactional logic, if needed, is supported by the application, not by the data engine
- the data engine is distributed by design, so it may scale up by adding commodity hardware

NoSQL Perl APIs ?

- Some NoSQL engines work as “services” behind a REST API, i.e. language-neutral
- Some NoSQL engines offer “connected” APIs ; they are provided with a low level client library, and sometimes with high level drivers for one or more dynamic *P** languages (but *P** is almost never *Perl*)
- Javascript is quickly gaining ground as a server-side development tool ; it's a strong challenger for *P** languages
- MongoDB is currently an exception : it offers a connected API that comes with *official* drivers for 13 languages, including a CPAN-packaged, conveniently documented Perl module



The document store data model

Attempted disambiguation

- The "*document database*" concept is far from new and may apply to very heterogeneous products/solutions
- So-called "unstructured" document management, sharing and replication platforms such as, say, IBM's *Lotus Notes*, EMC's *Documentum* or Microsoft's *Sharepoint* represent a well known kind of document databases
- Popular CMSs use SQL engines to store the documents
- XML DBMSs are real NoSQL document DBMSs (and in most cases notorious commercial failures)
- In the recently emerged NoSQL ecosystem (~2009), the "document store" concept is more restrictive

The “document store” in our scope

- The *document* is basically a consistent set of *key-value pairs*
- Each key is uniquely named in the document
- The value in each pair maybe either a scalar content or a sub-document (recursively)
- Each document belongs to a logical document set (*collection*, *database*, ...)
- Each document is uniquely identified in the document container
- Each document comes with its own structural metadata (allowing various structures in the same document set) and doesn't depend on any external reference (no foreign key, no referential integrity, no XML-like schema)

Document external représentation

Goodbye XML, welcome to JSON...

```
{
"First name" : "Jean-Marie",
"Last name"  : "Gouarné",
"email"      : [
     "jmgdoc@cpan.org",
     "jmg@genicorp.fr"
     ],
"Address"    : {
     "Country"     : "FR",
     "Street"      : "5 rue de la Forge",
     "City"        : "Paris",
     "Postal area" : "75017"
     }
}
```

Good news for Perl

A JSON document is (almost) a Perl hashref

```
$doc =  {
        "First name" => "Jean-Marie",
        "Last name"  => "Gouarné",
        "email"      => [
             "jmgdoc@cpan.org",
             "jmg@genicorp.fr"
             ],
        "Address"    => {
             "Country"     => "FR",
             "Street"      => "5 rue de la Forge",
             "City"        => "Paris",
             "Postal area" => "75017"
             }
        }
```

mongoDB

- Not the best NoSQL server for any purpose, but...
- One minute install for a single server with default config
- Powerful, resilient and reasonably easy to deal with in big data, read only environments
- Handle with care (or look for other products) in heavy write concurrency contexts (*note : stay tuned, improvements are coming*)
- Powerful and easy to use interactive javascript client shell
- Server side javascript engine
- Cool APIs for Perl and various other languages
- Probably the most friendly NoSQL DBMS for a Perl/SQL developer

The API

- Works in *connected* mode, so...
- A client application (ex: Perl) can get access to a MongoDB instance through one or more *connections*
- From an existing *connection*, the application may access one or more *databases*, i.e. logical data areas (not schemas)
- From a given *database*, the application can handle one or more *collections*
- A *collection* (as an API object) provides the most usual methods to create, update, delete or retrieve *documents*

The Perl API basics

Before coding

- Secure a local or remote MongoDB access (check it using the *mongo* shell)
- Install *MongoDB >= 0.503* from CPAN (beware : lot of dependencies)
- Use *mongoimport* to load data from flat files (no coding if the stuff is available in *CSV* or *JSON* format)
- Use the *mongo* shell to quickly explore the data
- Have a look at *MongoDB::Tutorial*
- Be prepared to frequent visits at *http://docs.mongodb.org* (and remember that the understanding of advanced code examples requires some Javascript reading skills)

Getting started

```
use MongoDB;

# establish a first connection
my $connection = MongoDB::MongoClient
          ->new('mongodb://localhost:27017');

# instantiate a DB object
my $db = $connection->get_database('demo');

# instantiate and use a collection
my $collection = $db->get_collection('hello');
my $id = $collection->insert(
     { author => "JMG", text => "Hello World !" }
     );
```

Retrieving documents

- For data set retrieval, the *find()* method returns a *cursor* that allows the application to iterate through the data set
- For single document retrieval, *find_one()* returns the (first) matching document as a Perl hash ref
- *find()* and *find_one()* arguments are *documents* (i.e. hash refs)
  - the first one is the condition; it may filter on one or more fields; regexs are supported
  - the second one is the list of fields required in the result set
- Main limit : no direct join with *find*/*find_one*; multi-collection queries imply procedures

Retrieving documents

Selecting on "field1" by value AND "field2" by regex, returning a dataset with only "field1" and "field4", sorted by "field1" descending. The loop iterates through the dataset for local processing.

```
$dataset = $collection->find(
     { field1 => $value1, field2 => qr/xxx/i },
     { field1 => 1, field4 => 1 }
)->sort({ field1 => -1 });

while ($doc = $dataset->next) {
     say "$doc->{field1} $doc->{field4}";
}
```

“$” operators

Filtering operators, if any, must be passed as special keys among the data.
The query below selects the places (city name + zip code) that belong to a restricted list of US states and whose population is greater than 100,000 (see the “zips” example in the MongoDB Manual for the data structure)

```
$states = ['NY', 'TX', 'IL', 'CA'];
$cond   = {
      pop   => {'$gt' => 100000},
      state => {'$in' => $states}
};
$fields = {city => 1, state => 1, pop => 1};
$sort   = {pop => -1, state => 1, city => 1, zip => 1};

$dataset = $cities->find($condition, $fields)->sort($sort);
```

Updating documents

Updating every document that matches the same conditions as in the previous example. The "multiple" option means "everyone" (and not only the first match). The code below adds a "comment" field to every document matching the same condition as in the previous query

```
$states      = ['NY', 'TX', 'IL', 'CA'];
$condition   = {
      pop         => {'$gt' => 100000},
      state       => {'$in' => $states}
};
$action      = {'$set' => {comment => "Large post office"}};

$cities->update($condition, $action, {multiple => 1});
```

mongoDB

More advanced examples

Aggregation, Geolocation

Aggregation

- Sophisticated aggregation framework for reporting and business intelligence needs
- Similar functionality to SQL “*group by*”
- Main method : *aggregate()* ; operates in pipe-line mode
- Powerful functionality for a NoSQL engine, but abashing grammar ; requires training...

Aggregation

SELECT state, SUM(pop) AS pop
FROM zips
GROUP BY state
HAVING pop > (10000000)
ORDER BY pop DESC

```
my $dataset = $c->aggregate([
    {'$group' =>
        {'_id' => '$state', 'totalPop' =>
            {'$sum' => '$pop' }
        }
    },
    {'$match' =>
        {'totalPop' => { '$gte' => 10000000 }}
    },
    {'$sort'  => { totalPop => -1 }}
]);
```

Geospatial queries

- The MongoDB engine may recognize some particular fields as 2D coordinates
- These particular fields may be used in *find()* queries in order to select documents according to their geographical position
- They allow filtering on
  - distance from a given point ;
  - location within a given polygon (array of points)
- They allow ranking according the geographical distance from a given location

Geospatial queries

```
{
     nom_lieu        : "Bruxelles",
     code_postal     : "1000",
     nom_region      : "Bruxelles-Capitale",
     position        : [ 50.8466, 4.3528 ],
     …
}
```

Selecting the 20 nearest locations from a given point according to the "position" field (assuming document structures similar to the above)

```
my $places = $collection->find(
     {position => {'$near' => [$x, $y]}}
    )
    ->limit(20);
```

mongoDB

Server-side code execution

A Perl/Javascript affair

- A Perl program can provide Javascript functions to be executed in the *server* space
- The client program just submit the code and gets the result; there is no intermediate network traffic between the client and the server
- This feature may be used either to call functions persistently stored in the database or to execute application-provided functions
- The JS is submitted through the *eval()* database method, and is executed by the server in the context of the calling database

A Perl/Javascript affair

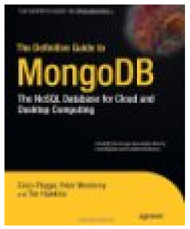
Submitting a Perl-generated Javascript function and processing the result

```
$js = 'function(arg1, arg2) { ... }';
$response = $database->eval(
     $js, \@arguments
);

if ($response->{ok}) {
     process($response->{result};
} else {
     say "Server side execution failure";
}
```

Recommended readings

- **CPAN** (*http://search.cpan.org/dist/MongoDB*)
- **The little MongoDB Book** – Karl Seguin (*http://openmymind.net*)
- **MongoDB Doc Project** (*http://docs.mongodb.org*)
- **MongoDB, the definitive guide** – K. Chodorow M. Dirolf, *O'Reilly*
- **MongoDB in action** – Kyle Banker, *Manning*
- **The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing** – E. Plugge, T. Hawkins P. Membrey, *APRESS*



Questions ?