

# An Open Source NoSQL solution for Internet Access Logs Analysis

A practical case of why, what and how to use a NoSQL Database Management System instead of a relational one



*José Manuel Ciges Regueiro <jmanuel@ciges.net> - Student of the V Master on Free Software Projects Development and Management 2011-2012*



# Author's description

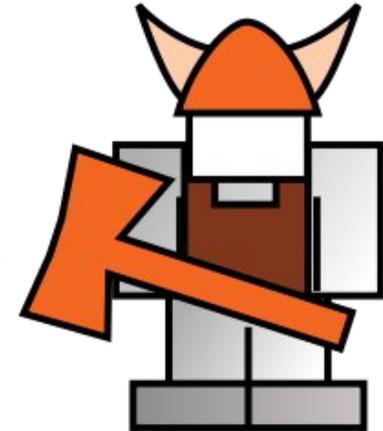
- Spanish systems engineer who has been working at PSA Peugeot Citroën for the last 9 years
- My department provides support for Open Source servers on Unix
  - Apache web server, MySQL, PHP, Tomcat, MediaWiki ...
  - We work with around 450 servers
- The IT department at PSA employs 2.600 people in 26 different countries and has around 10.000 servers including Unix, Windows, z/OS, VMS, Tandem/Guardian ...
- I'm also:
  - Father
  - Internet, GNU/Linux & Free Software fan boy
  - Always looking for something :-)

# Why NoSQL?

- The term “NoSQL” exists from 1998 and is used to designate DBMS who don't use SQL
- Problems like statistical data analysis, log information, web page indexing, geographical information management ....
  - manage huge amounts of data
  - use distributed data between very many machines
  - distribute also the **operations on the data**
  - need a fault tolerant architecture
  - **read & write performance are critical**. Relations are not really important, it's preferred to duplicate data
  - May not give full ACID guarantees
- Who uses it? Google, Facebook, Amazon, Twitter ...



- Our goals were
  - Acquire knowledge on the subject
  - Identify what NoSQL technologies could be useful for PSA
  - Get one or more tests cases and build a demo
  - Offer a NoSQL solution to development team
- The work plan has been
  - Read, read & read about current Open Source NoSQL solutions
  - Choose one for the use case of **Internet Access Log management**
  - Develop a little API, scripts and design a schema to **compare NoSQL vs MySQL**
  - Compare them



# Types of NoSQL DBMS

NoSQL databases are categorized according to the way they store the data

- **“Document-oriented databases”** : The element of data is called “document”. Each document can have a different number of fields

MongoDB, CouchDB

- **“Key-value databases”** : Data is stored as key-value pairs. A value can be of any data type or object.

Cassandra, Membase, Redis, Riak

- **“Graph databases”** : Oriented towards data whose relations are well represented with a graph-style (social relations, maps, network topologies ...)

Neo4J, FlockDB

- **“Tabular/Columnar databases”** : data is organized in columns. Each row has one or more values for a number of possible columns.

HBase, Cassandra

# Analyse of Internet access logs

- Storage and analysis of the logs saved by the proxies who give access to the Internet
- Each hit has information like: Date and time of access, user name, accessed URL, size in bytes ...
- Questions we want to answer:
  - Which are the most visited pages?
  - And the most visited per month? And last week?
  - Which users spend more time online?
  - Which are the 100 users whose traffic volume is greater?
  - What is the average daily volume of traffic from the corporate network to the Internet?
- Volume estimation by month (for 70.000 users)
  - Data size: between 150 and 300 GB
  - Log entries number: between 650 million and 1.700 million



In a year we could reach a stored volume of 3.600 GB for 20 billion log entries



# Searching for a NoSQL solution

- We should be able to answer the following questions:

- What type of data will be handled?

Can this data be naturally organized in associative Arrays? Or in key-value pairs? Is it data which will fit in a XML or similar structure?

- Do we need transactions?

- Do we need to use “Map Reduce”?

- And when reviewing the different options:

- Is the latest version considered stable?

- Does it have commercial support?

- What is the learning curve?

- Is good documentation available? Is there an active community?





# Description of our data

- Access logs generated by several HTTP proxies:
- Two different type of records: records from FTP access and from the rest (mainly HTTP)
  - For each FTP access we will save: IP, user, date and time, accessed domain, URI, Size
  - For each NonFTP access: IP, user, date and time, HTTP method used, protocol, accessed domain, URI, return code, size
- The following statistical reports will be created:
  - Number of hits and volume of data transferred **by Internet domain, daily and monthly**
  - Number of hits and volume of data transferred by user, daily and monthly





# Definition of our needs

- The data:
  - data is composed by records with multiple fields
    - **document-oriented database or tabular**
  - records are unrelated to each other
  - each entry is stored in a log table as it grows indefinitely
  - accesses to the database are **mostly writing**
- The reports:
  - the list of queries sent by our application is known
  - **Map Reduce is desired**
  - each access means a change in daily and monthly access totals by domain and user for having **real-time** information



## Definition of our needs (2)

- We don't need:
  - master-master replication (proxies in different geographic areas manage accesses from different users)
  - support for multiple versions
  - real data consistency
  - Transactions
- Also, the chosen product must be:
  - Open Source
  - Ready for production environments
  - With **professional support**

# Choosing between several NoSQL solutions

- If we discard the databases that hold data in memory, as key-value pairs and graphs we are left with: MongoDB, CouchDB, Cassandra, HBase and Riak
- **At last I have chosen MongoDB**
- Why not CouchDB:
  - We don't need versioning
  - It's not very mature (latest version is 1.2.0 and has changes that make it incompatible with the previous versions)
  - To exploit the data it is necessary to define views previously
- Why not Riak:
  - It has two versions, one open source and a commercial one with multi-site replication
- Why not Cassandra:
  - Too complex
- Why not HBase:
  - Too complex



Meets all the requirements stated at the beginning

- **document-oriented** and very flexible in structuring the data (uses JSON)
- has support for **Map-Reduce**
- stable and considered production ready (current version is 2.2)
- has a complete website with **extensive documentation and comprehensive guides**
- professionally supported. Support is given by the same company that developed the product, 10gen.
  - very active, they are present in many conferences and lectures (FOSDEM 2012, by example)
- comparatively this product does not seem too complex
- there are native drivers for multiple languages made by 10gen
- Open Source, of course :-)





# Schema Design for Internet Access Log control

## Equivalent MySQL database

- **Access Logs:** FTP connections are stored in a different table than the Non FTP (mostly HTTP)
- Two **totals** are stored per user and domain by month: number of access and volume in bytes downloaded. Then, for **each month we have two tables**, one with the users information and a second one with domains information.

Non_FTP_Access_Log	
◇	clientip
◇	user
◇	datetime
◇	method
◇	protocol
◇	domain
◇	uri
◇	return_code
◇	size

FTP_Access_Log	
◇	clientip
◇	user
◇	datetime
◇	method
◇	domain
◇	uri
◇	size

Users_Monthly_Report_201206	
◇	user
◇	nb
◇	volume

Domains_Monthly_Report_201206	
◇	domain
◇	nb
◇	volume



# Schema Design for Internet Access Log control

## MongoDB

- Data is grouped into “collections” (as tables) and each element of data is called a “document”.
- Unlike relational databases **there is no need to define an structure**. Each document could have a **different number of fields**, and also contain other documents.

### NonFTP\_Access\_Log

```
{
  "userip": string,
  "user": string,
  "datetime": MongoDate,
  "method": string,
  "protocol": string,
  "domain": string,
  "uri": string,
  "return_code": integer,
  "size": integer
}
```

### FTP\_Access\_Log

```
{
  "userip": string,
  "user": string,
  "datetime": Date,
  "method": string,
  "domain": string,
  "uri": string,
  "size": integer
}
```

### Users\_Monthly\_Report\_201204

```
{
  "_id": "Userid"
  "Nb": integer,
  "Volume": integer,
  "Daily": {
    "0": {
      "Nb": integer,
      "Volume": integer
    },
    "1": {
      "Nb": integer,
      "Volume": integer
    },
    "2": {
      "Nb": integer,
      "Volume": integer
    },
    "3": {
      "Nb": integer,
      "Volume": integer
    },
    ...
    "30": {
      "Nb": integer,
      "Volume": integer
    },
  },
},
}
```

*MongoDB schema shown as pseudo code*

# Comparative of a MySQL based solution vs MongoDB

We are going to:

- **Develop code** to fill them with “fake but realistic” data
  - 70.000 users
  - 70.000 IPs
  - 1.300.000 Internet domains
  - 90.000.000 of Non FTP log entries
  - 4.500.000 of FTP log entries
- **Define a battery of tests** to compare the performance of both solutions
  - MongoDB 2.2.0
  - MySQL 5.0.26 with MyISAM tables



# PHP classes developed

I have developed three PHP classes:

- “**RandomElements**” class: with functions like getRandomDomain(), getRandomFTPMethod() ... which are used to generate the random elements of data
- “**MongoRandomElements**” and “**MySQLRandomElements**” classes, which are children classes of the previous one and have added functions to work with each database management system.
  - The interface for these two classes is the same
  - These classes are used by almost all the tests scripts

They have functions to:

- Save a random user in the database
- Create lists of random domains, IPs and users and save them in tables/collections
- Verify if a user exists in the list of random users
- Get one random user/domain/IP from the list of created users/domains/IPs
- Create a new log entry getting the (random) elements needed and save it into the database
- ...



# PHP classes developed (2)

## Sample code:

```
$mre = new MongoRandomElements();

$mre->createUsers(70000);

$mre->createIPs(70000);

$mre->createDomains(1300000);

// Example data for April

$start = mktime(0,0,0,4,1,2012);

$end = mktime(23,59,0,4,30,2012);

for ($i = 0; $i < 30000000; $i++) {

    $log = $mre->getRandomNonFTPLogEntry($start, $end);

    $mre->saveRandomNonFTPLogEntry($log);

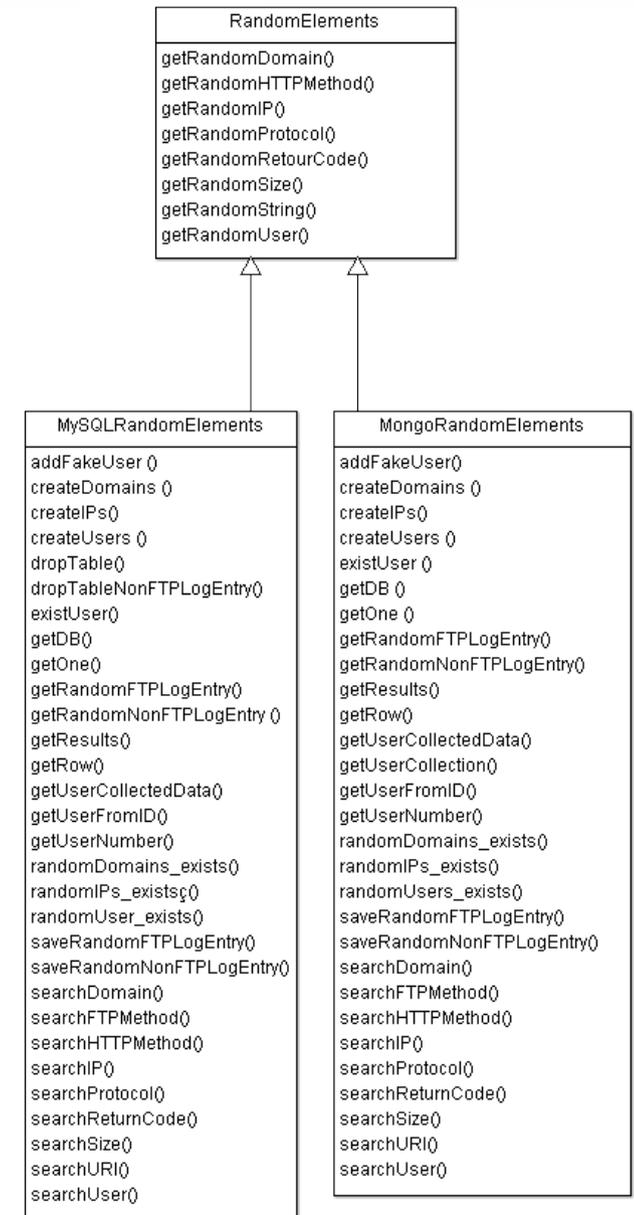
}

for ($i = 0; $i < 1500000; $i++) {

    $log = $mre->getRandomFTPLogEntry($start, $end);

    $mre->saveRandomFTPLogEntry($log);

}
```



UML diagram



# Insertion tests results

We ran 10 tests for inserting data in MongoDB and MySQL (20 PHP scripts)

	MongoDB	MySQL
70.000 users	3s	12s
70.000 IPs	3s	12s
1.300.000 domains	58s	4m 36s
70.000 unique users with indexes	23s	28s
70.000 unique IPs with indexes	22s	31s
1.300.000 unique domains with indexes	8m27s	14m13s
1.000.000 log entries	12m7s	26m14s
5.000.000 log entries	1h03m53s	2h10m54s
10.000.000 log entries	1h59m11s	3h27m10s
30.000.000 log entries	5h55m25s	10h18m46s

# Multiuser concurrent tests

These tests will simulate simultaneously accessing users

- The request are made to **PHP scripts available via web**
- The load tests were done with the open source tool **JMeter** and the graphical representation with **R**

Six scripts were developed, which will perform the following tests for MongoDB and for MySQL

- Search and show data for a random user (**read test**)
- Add a random user (**write test**)
- Search and show data for a random user or add a random user (**read & write test, 80% of times will read and 20% of times will write**)

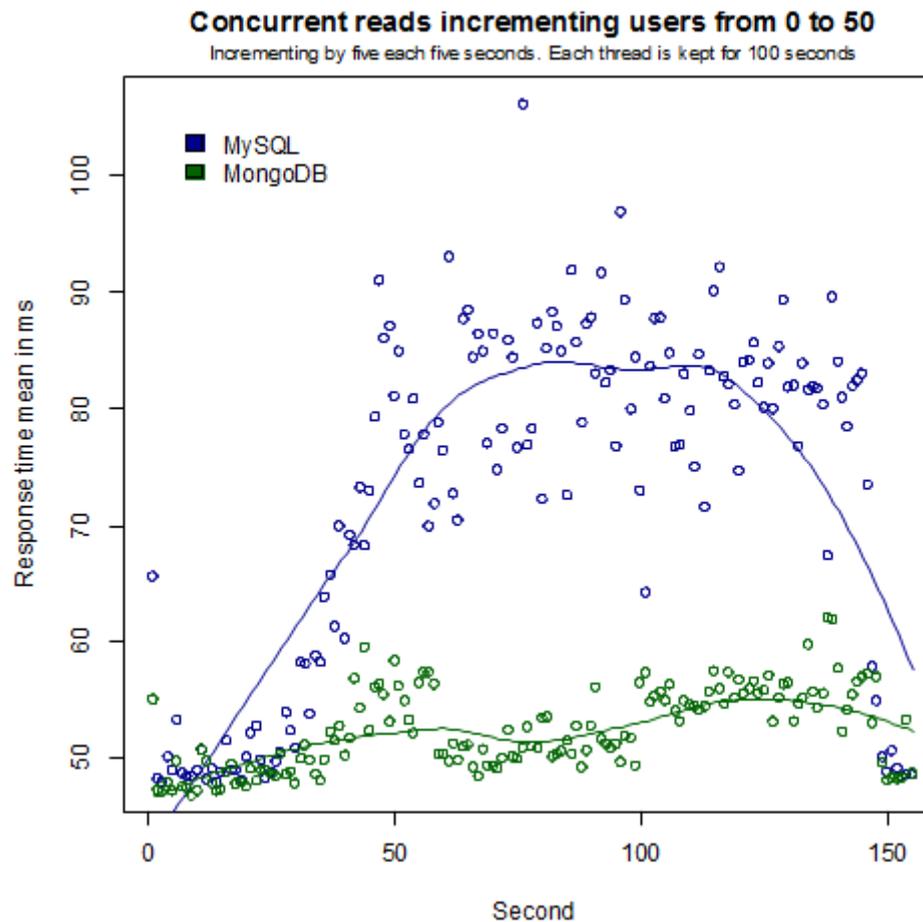
We will simulate two scenarios

- An incrementing load from 0 to 50 users rising by five
- A load of 50 users sending all queries from the beginning.

# Concurrent reads tests results

Incrementing users from 0 to 50 (each thread will be kept for 100 seconds)

	Samples	Med	Min	Max	Std. Dev.	Throughput	KB/sec
<b>MongoDB</b>	112.165	<b>48ms</b>	43ms	3.090ms	18,72ms	728,3 q/s	184,99 kb/s
<b>MySQL</b>	81.179	<b>67ms</b>	45ms	3.140ms	40,92ms	528 q/s	134,08 kb/s

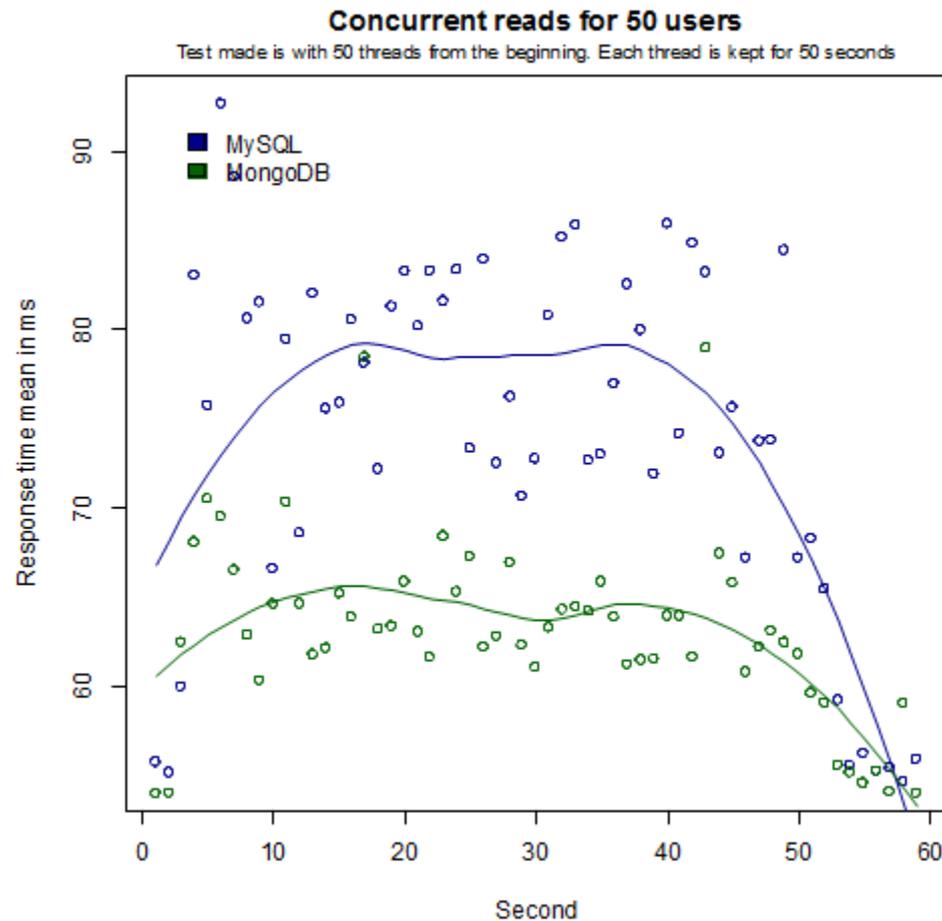




# Concurrent reads tests results (2)

50 users from the beginning (each thread will be kept for 50 seconds)

	Samples	Med	Min	Max	Std. Dev.	Throughput	KB/sec
<b>MongoDB</b>	37.497	<b>54ms</b>	50ms	5.419ms	161,04ms	635,4 q/s	122,88 kb/s
<b>MySQL</b>	32.273	<b>62ms</b>	52ms	5.136ms	156,90ms	547,9 q/s	114,54 kb/s

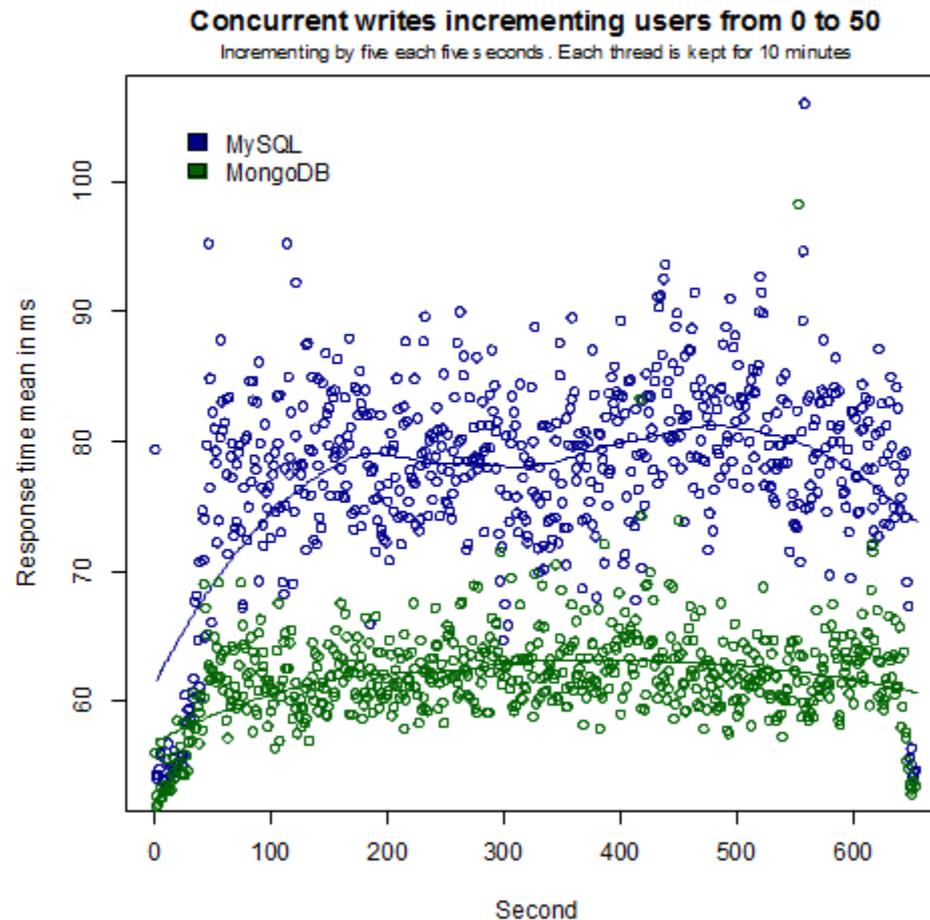




# Concurrent writes tests results

Incrementing users from 0 to 50 (each thread will be kept for **10 minutes**)

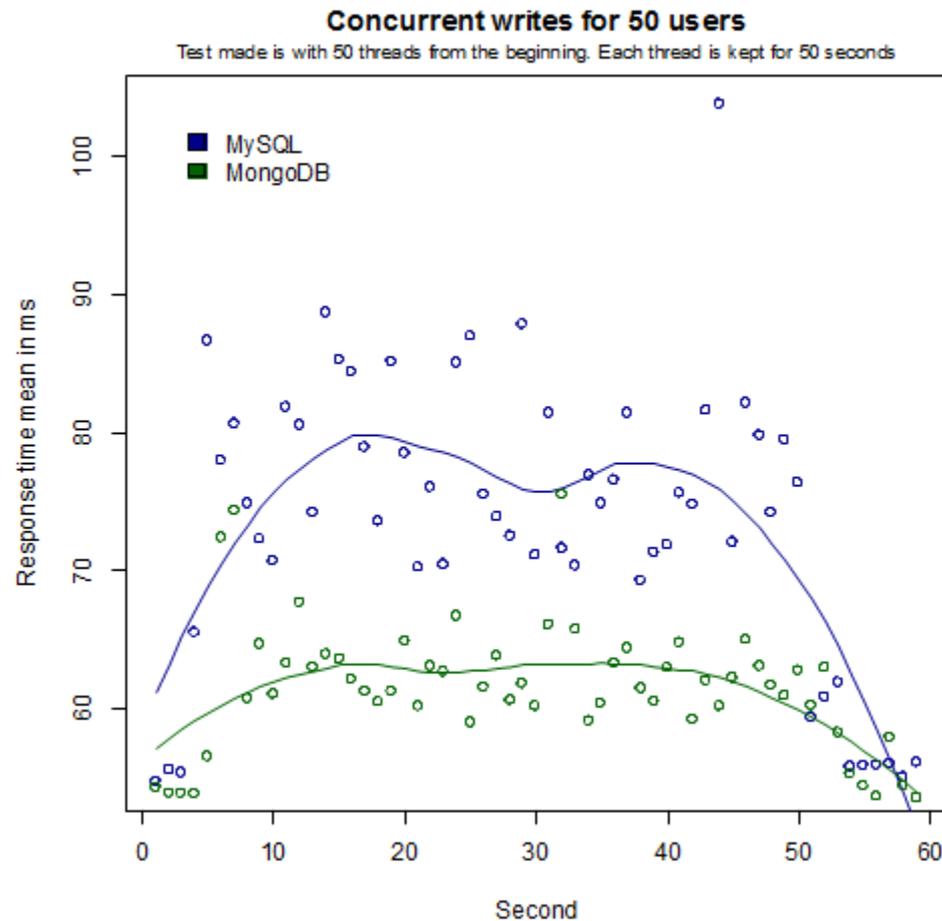
	Samples	Med	Min	Max	Std. Dev.	Throughput	KB/sec
<b>MongoDB</b>	464.853	<b>54ms</b>	49ms	3.105ms	27,71ms	710,9 q/s	148,67 kb/s
<b>MySQL</b>	383.700	<b>70ms</b>	51ms	4.105ms	25,58ms	586,7 q/s	122,64 kb/s



# Concurrent writes tests results (2)

50 users from the beginning (each thread will be kept for 50 seconds)

	Samples	Med	Min	Max	Std. Dev.	Throughput	KB/sec
<b>MongoDB</b>	37.497	<b>54ms</b>	50ms	5.419ms	161,04ms	635,4 q/s	132,88 kb/s
<b>MySQL</b>	32.273	<b>62ms</b>	52ms	5.136ms	156,90ms	547,9 q/s	114,54 kb/s

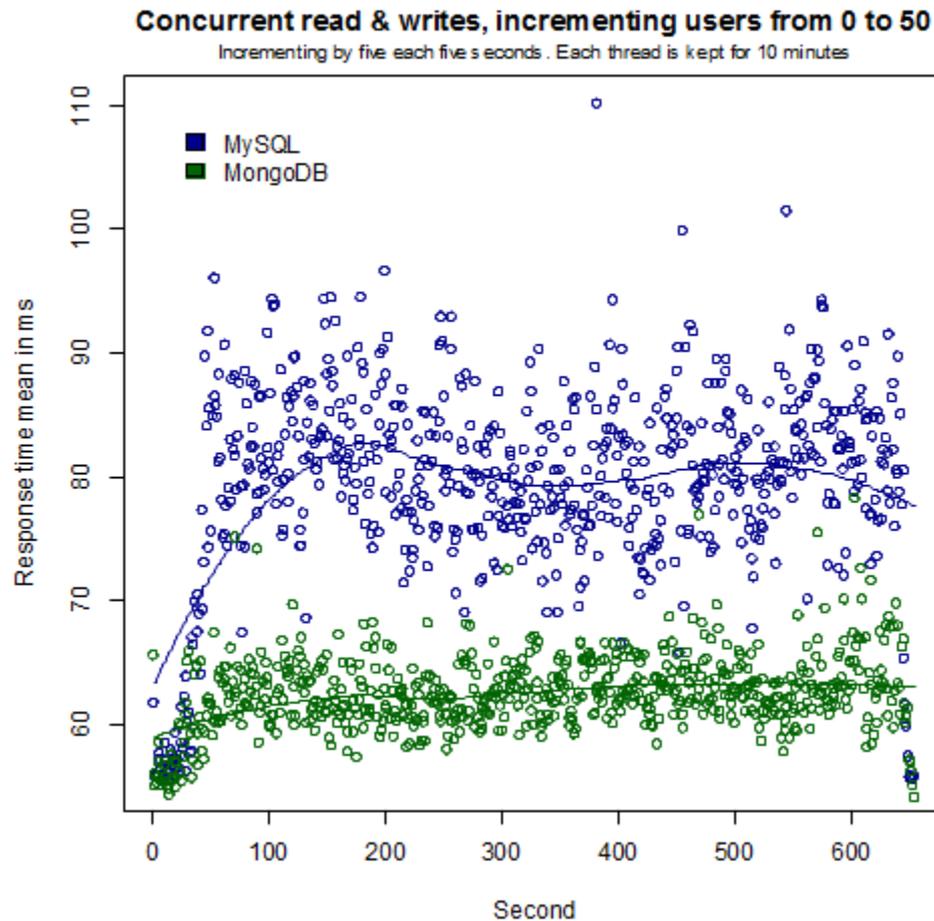




# Concurrent reads & writes tests results

Incrementing users from 0 to 50 (each thread will be kept for **10 minutes**)

	Samples	Med	Min	Max	Std. Dev.	Throughput	KB/sec
<b>MongoDB</b>	462.740	<b>55ms</b>	48ms	3.111ms	26,24ms	707,7 q/s	173,45 kb/s
<b>MySQL</b>	373.484	<b>71ms</b>	52ms	3.152ms	27,98ms	571,1 q/s	139,91 kb/s

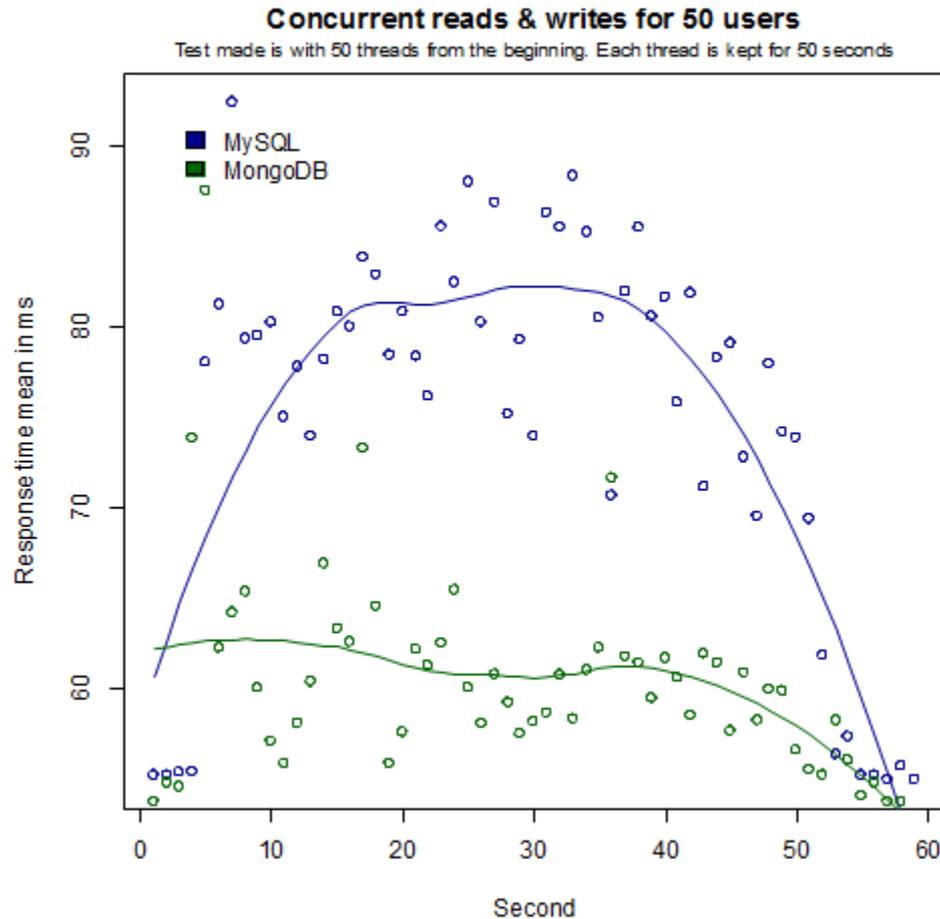




# Concurrent reads & writes tests results

50 users from the beginning (each thread will be kept for 50 seconds)

	Samples	Med	Min	Max	Std. Dev.	Throughput	KB/sec
<b>MongoDB</b>	39.096	<b>54ms</b>	50ms	4.753ms	142,72ms	665,0 q/s	162,93 kb/s
<b>MySQL</b>	31.272	<b>64ms</b>	52ms	5.964ms	176,64ms	530,0 q/s	129,8 kb/s





# Data analyse (aggregation) read tests

- This tests are made to compare the aggregation capabilities of both database management systems.
- The queries have been done over 90 million of log records
  - Which are the 10 most visited domains and how many visits has each one?
  - Which are the 10 most visited domains in the second half of June?
  - Which are the 10 users that have more Internet accesses?
  - What is the average Internet traffic for June?

	MongoDB	MySQL
10 most visited domains with visit totals	13m13s	2m37s
10 most visited domains in the second half of June	52m39s	17m43s
10 users with the most Internet accesses	24m02s	3m53s
Average Internet traffic for June	12m05s	2m42s

# Example of MongoDB's aggregation framework

Which are the 10 most visited domains in the second half of June?

- First, we get the minimum value of the 10 highest visits per domain

```
$mre = new MongoRandomElements("mongodb", "mongodb", "localhost",  
"InternetAccessLog");  
$start = new MongoDate(strtotime("2012-06-15 00:00:00"));  
$end = new MongoDate(strtotime("2012-06-30 23:59:59"));  
$min_value = $mre->getOne(array(  
    array('$match' => array('datetime' => array( '$gt' => $start, '$lt' =>  
$end ))),  
    array('$group' => array('_id' => '$domain', 'visits' => array( '$sum' => 1  
))),  
    array('$group' => array('_id' => '$visits')),  
    array('$sort' => array('_id' => -1)),  
    array('$limit' => 10),  
    array('$sort' => array('_id' => 1)),  
    array('$limit' => 1),  
), "NonFTP_Access_log");
```

- Then, we obtain all the domains with at least that value for the number of visits

```
$data = $mre->getResults(array(  
    array('$match' => array('datetime' => array( '$gt' => $start, '$lt' =>  
$end ))),  
    array('$group' => array('_id' => '$domain', 'visits' => array( '$sum' => 1  
))),  
    array('$match' => array('visits' => array( '$gte' => $min_value)))  
), "NonFTP_Access_log");  
  
foreach($data as $doc) { print_r($doc); }
```

# Conclusions and last words

## Write performance:

- **MongoDB is faster in pure write performance**
- For continuous simple writings MongoDB is from 2 to 4 times faster. For high numbers simple writing performance is the double of MySQL
- In concurrent writes MongoDB is faster (15% and 30% in our tests)

## Read performance:

- **MongoDB is faster in pure read performance**
- In concurrent reads MongoDB is faster (15% and 40% in our tests)

## Aggregation performance:

- Here MySQL wins over MongoDB's aggregation native framework. **MySQL is much faster in aggregating data**, 3 to 6 times faster for the 4 tests we have done

**MongoDB is more scalable**, meaning that when the user load increases the response time keeps stable.

For intensive reading and writing data operations MongoDB is a better option than MySQL when no relations nor aggregation queries performance are important and the data reading/writing performance is critical.



# Initial planning and actual time spent on each task

The initial estimated work charge was 300 hours. The actual time spent has been of more of 400 hours divided as follows:

Tasks	Time spent
Study of NoSQL articles & books	65 h
MongoDB installation, configuration, package creation & updates	58 h
Development of a schema for Internet Access Log	20 h
Scripts development (PHP, SQL, JavaScript, MySQL stored procedures)	68 h
Load tests	75 h
Documentation (memory, posts on ciges.net & presentation)	93 h
Incidents analyse & resolution	18 h
Planning, coordination & communication	43 h
<b>Total</b>	<b>440 h</b>

To complete this work the following should be done

- Add tests with a multi-machine configuration using **sharding**

Also other future lines of work could be:

- Test map-reduce operations with **Hadoop integration**
- Test map-reduce operations with **V8 JavaScript engine**
- Repeat the tests with a **huge quantity of data**



# Contributions to the community



The contribution to the community is documentation and source code.

All **source code (PHP classes, scripts and configuration files)**, **documentation and detailed instructions to repeat the tests** are available on

- Github repository “Ciges / internet\_access\_control\_demo”
- My personal web page <http://www.ciges.net>, which has been created and contains a series of posts which summarize this work (in Spanish)

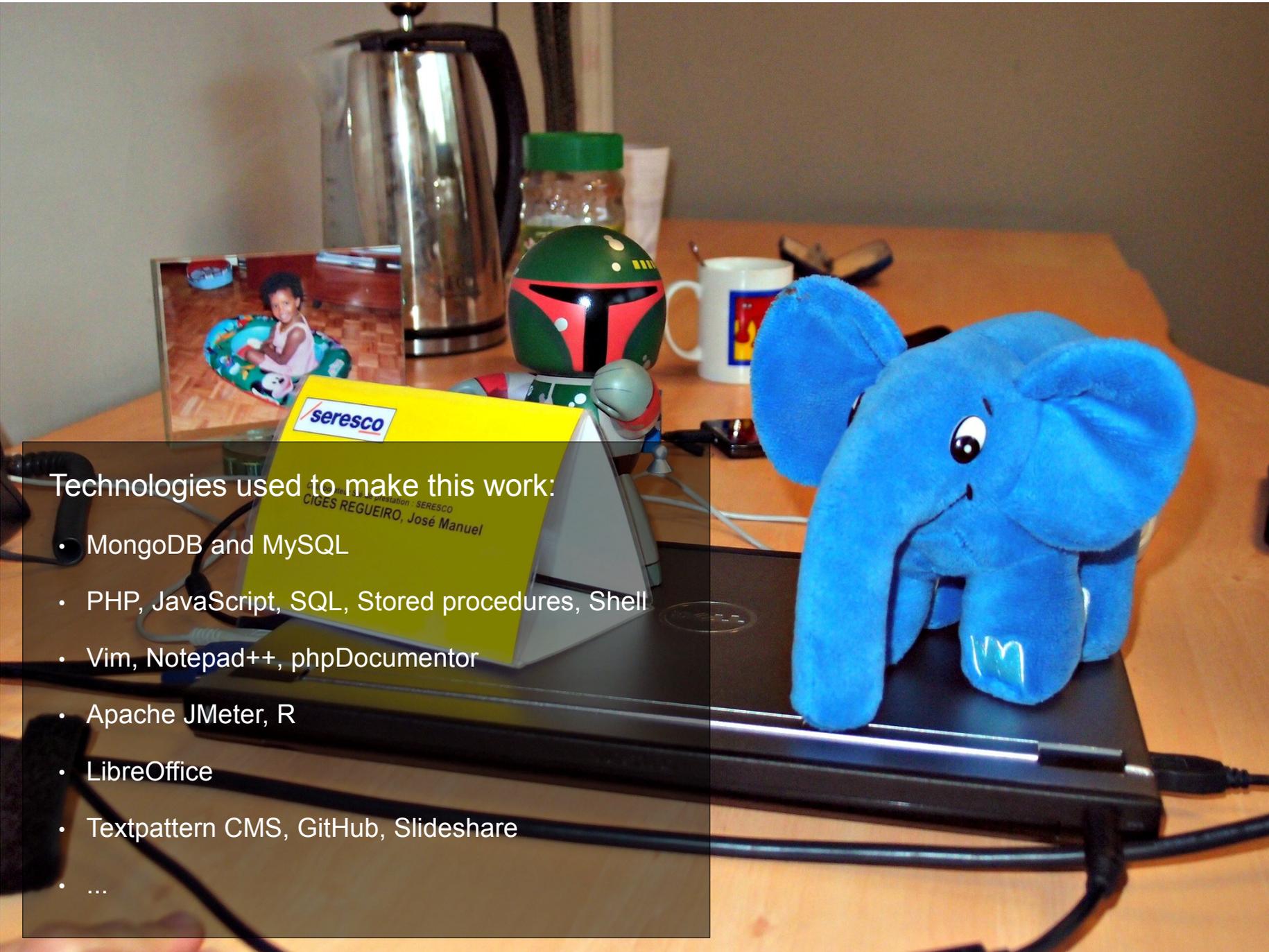
Contributions to the **Wikipedia**

- Rewriting of English articles: “MongoDB”, “CouchDB” and French “MongoDB”
- Minor edition on other articles like English “CAP theorem”, “NoSQL”, “Textile (markup language)”, “Apache Cassandra”

Questions **opened and answered on Stack Overflow** (and also in MongoDB's JIRA)

- “Map Reduce with MongoDB really, really slow (30 hours vs 20 minutes in MySQL for an equivalent database)”
- “Simple tool for web server benchmarking?”
- “Simultaneous users for web load tests in JMeter?”

# Technologies used



Technologies used to make this work:

- MongoDB and MySQL
- PHP, JavaScript, SQL, Stored procedures, Shell
- Vim, Notepad++, phpDocumentor
- Apache JMeter, R
- LibreOffice
- Textpattern CMS, GitHub, Slideshare
- ...

# That's all Folks! Any questions?



This presentation, all the articles and documents have the licence Creative Commons Attribution-ShareAlike 3.0 Unported.



The source code has the licence GNU GPL 3.0



Most of clip arts have been taken from Open Clip Art Library web [openclipart.org](http://openclipart.org)