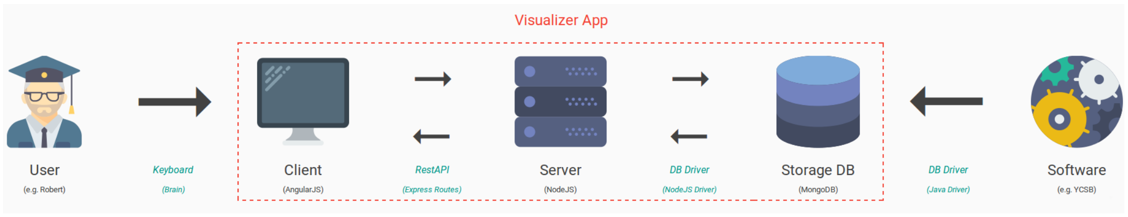
README.md

Web Dataset Visualizer

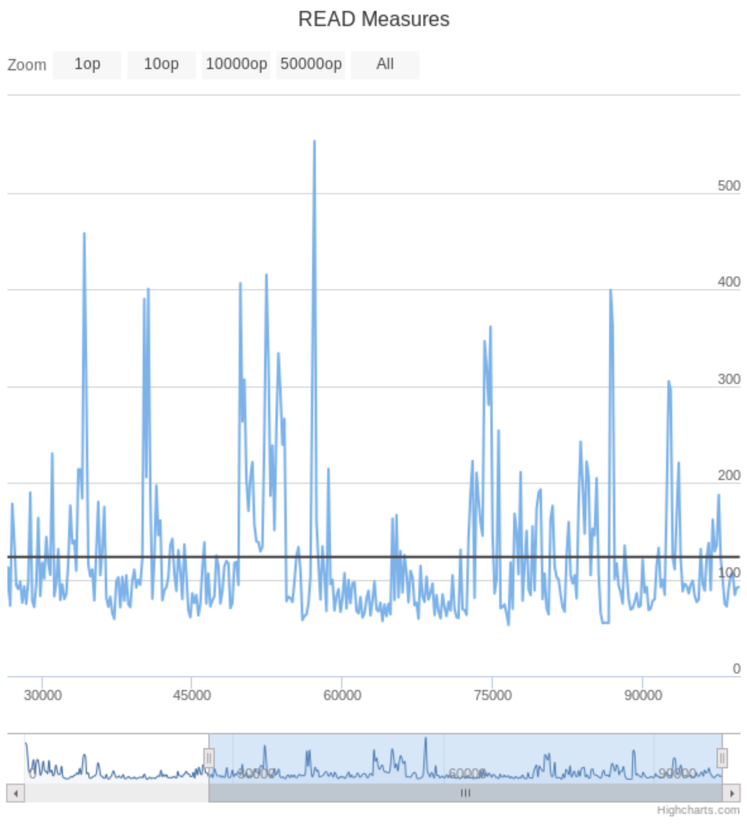
Presentation

Originally design for Yahoo! Cloud Serving Benchmark, this web application displays measures stored in a MongoDB database. It could be used for any kind of data and series type but is mostly used for benchmark measures visualization.

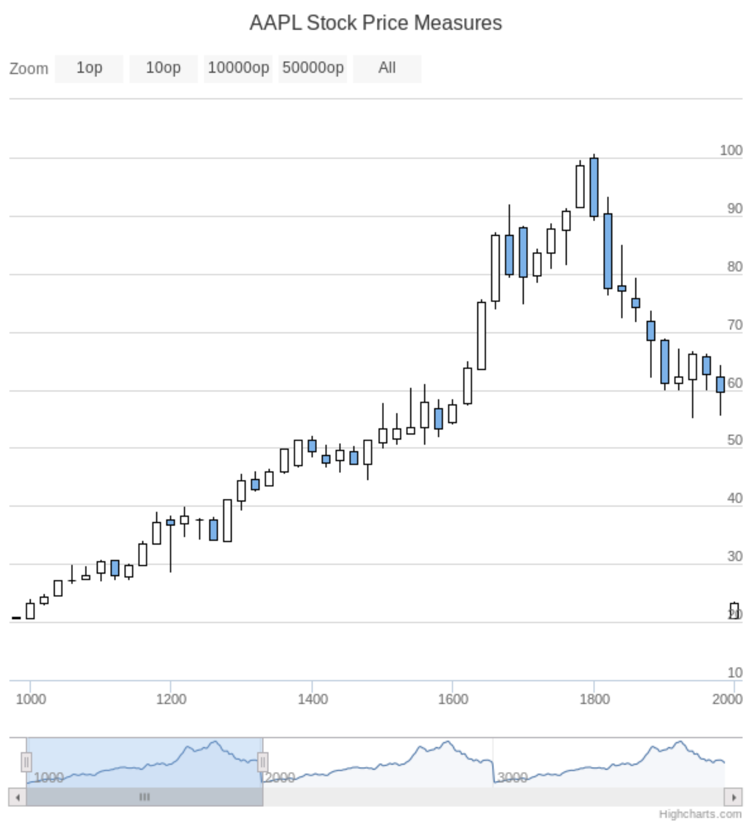


Place in the overall project architecture

It supports large dataset (millions of points) and achieves view optimization to display them efficiently. Here's some chart examples:



XY chart example



Boxplot example

This application currently has a full YCSB support, you can launch benchmark using YCSB parameters and make your own workload with a Web UI.

For now, only XYplot and Boxplot are supported. However, implementing a new series type is very easy! See Create your own conversion adapter

Getting started

Core programs

WARNING: We assume that nodejs and npm are already installed on your machine

You will need Java 8 and MongoDB to use this application :

```
sudo apt-get install openjdk-8-jre mongodb
```

YCSB specific requirements

DB program

You will need a DB to benchmark, we are going to use memcached :

```
sudo apt-get install memcached
```

YCSB supported version

You will also need YCSB "visualisation" version.

For now, the custom release will be included in this repo. In the future, we hope YCSB will accept our pull-request.

Configuration

Storage database

You can configure your MongoDB URL in the *config/database.js* file.

Here's the general config file with placeholders:

```
var urls = {
    localUrl: 'mongodb://localhost/<my_db_name>',
    remoteUrl : 'mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]][/[database][
};

module.exports = {
    url: urls.<my_url_attribute>
};
```

See MongoDB Documentation for connection string format explainations.

Local configuration example

Here's an example of a local configuration:

```
var urls = {
    localUrl: 'mongodb://localhost/dbMeasurements', // our DB name is dbMeasurements
};

module.exports = {
    url: urls.localUrl // we export the local url for the application
};
```

Remote configuration example

If you want to have a remote instance you need to :

- authorize your remote MongoDB instance to accept remote connections.
- adapt the *remoteUrl* attribute
- export *urls.remoteUrl*

This is an example of a remote configuration:

```
var urls = {
    remoteUrl : 'mongodb://node:nodeuser@mongo.onmodulus.net:27017/uwO3mypu'
};

module.exports = {
    url: urls.remoteUrl // we export the remote url for the application
};
```

YCSB & benchmarked database

Some more configuration are available in the *config/system.js* file.

```
var ycsbRoot = "/home/titouandocuments/ycsb-web-app/ycsb-0.11.0-custom-release/";

module.exports = {
    countersCollectionName: 'counters',

    ycsbExecutable: ycsbRoot + 'bin/ycsb',
    ycsbPythonExecutable: ycsbRoot + 'bin/ycsb',
    useBindingFile: true,
    ycsbBindingsFile: ycsbRoot + 'bin/bindings.properties',
    workloadFolder: ycsbRoot + 'workloads/',
    dbDumpsFolder: '/home/titouandocuments/ycsb-web-app/public/dumps/'
    evaluationsLocation: '/home/titouandocuments/ycsb-web-app/public/evaluations/'
};
```

You will find information on these variables in the file's comments.

Client configuration

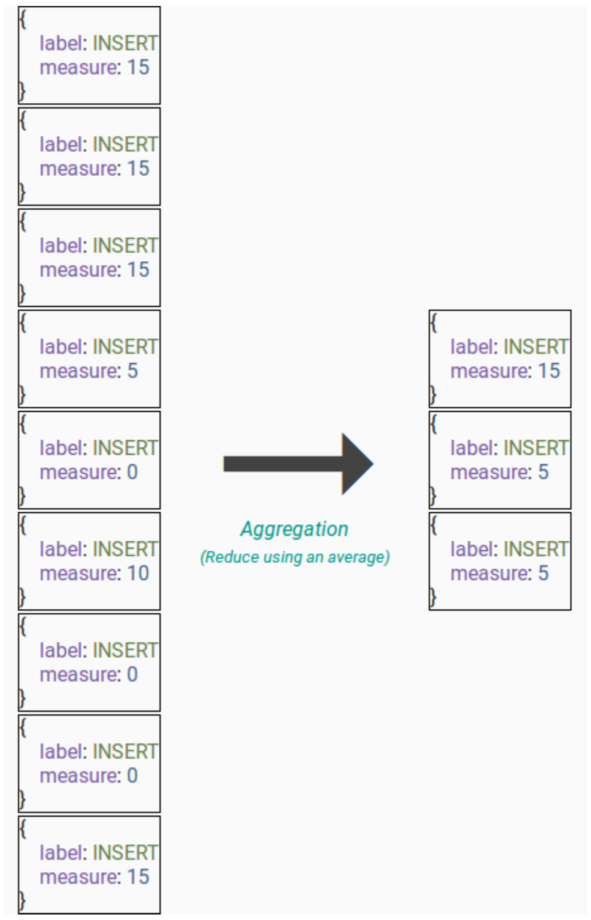
You might want to configure the charts view, here's a section for you. This configuration is optional for YCSB users.

Chart view optimization

If you want to make a large number of measure, our client won't be able to display all measures. Our client has an automatic aggregating process to overcome the problem.

Explainations

MongoDB is grouping value and making averages to optimise the view and make NodeJS serve results faster. This grouping process reduces measurements' precision. For example, you would get 3 buckets of average each based on 3 successive values instead of getting 9 measures:


{ label: INSERT measure: 15 }
{ label: INSERT measure: 15 }
{ label: INSERT measure: 15 }
{ label: INSERT measure: 5 }
{ label: INSERT measure: 0 }
{ label: INSERT measure: 10 }
{ label: INSERT measure: 0 }
{ label: INSERT measure: 0 }
{ label: INSERT measure: 15 }
Aggregation
(Reduce using an average)
{ label: INSERT measure: 15 }
{ label: INSERT measure: 5 }
{ label: INSERT measure: 5 }

Simple illustration of MongoDB aggregation process

The better your computer and browser are, the higher you can set the *MAX_POINTS* value and the smaller these buckets will be. You will have a more precise dataset.

Label definition

In order to generate a chart, you need to declare the different label you will have in your storage database and which Highcharts series type you want to use for this label.

For example, we have these documents in your storage database:

```
[
    {
        label: "INSERT",
        num: 0,
        measure: 4854
    },
    {
        label: "INSERT",
        num: 2,
        measure: 1254
    },
    {
        label: "CLEANUP",
        num: 0,
        measure: 105
    },
    {
        label: "INSERT",
        num: 3,
        measure: 45
    },
    ...
]
```

We need to declare the "INSERT" and "CLEANUP" label like that:

```
...
$scope.labelTypeMap = {
    "INSERT" : "line",
    "CLEANUP" : "line",
};
...
```

The visualization view will know display two charts, a cleanup and an insert one and automatically filled it with the corresponding data.

Variables' location

The variable is located in *public/stats/stats.controller.js*.

```
/** CONFIGURATION VARIABLES **/
$scope.MAX_POINTS = 20000; /* maximal number of points you can get from MongoDB */
$scope.showAverage = true; /* show average series or not */
$scope.labelTypeMap = {
    "INSERT" : "line",
    "READ" : "line",
    "UPDATE" : "line",
    "READ-MODIFY-WRITE" : "line",
    "CLEANUP" : "line",
    "SCAN" : "line",
};
```

Deployment

NodeJS server modules installation

Go to the root folder of the application where the file *package.json* is located and execute:

```
npm install
```

Client modules installation

Go to the *public* folder where the file *package.json* is located and execute:

```
npm install
```

Execution rights

Be sure to have execution rights on the app folder otherwise launching benchmarks won't work!

Launch it !

You need to start the NodeJS server:

```
node --max-old-space-size=16384 server.js
```

Then go to http://localhost:5555 !

NOTE: adapt the *max-old-space-size* relying on your machine performances.

What about not using YCSB ?

At first, our vizualizer was made for YCSB, but during the development we thought it would be great for it to support any kind of benchmarking software or more generally every software that output data.

These following points explain how to use our visualizer without YCSB.

MongoDB Population

First, your software should be able to populate a MongoDB database.

Each benchmark, group of measurement should be on a separate collection. Inside those collections, your documents have to respect the following scheme :

- a string label *label* (in our case it is the operation type INSERT, UPDATE, ...) which will separate your graphs
- a unique number *num* which will be use to sort your entries
- an object *measure* which is your measure

This last attribute of your documents can be at least whatever you want. This can be an object with fields, a single value, an array, etc. We will see how we handle this on the client side.

Example: our YCSB scheme

```
[
    {
        label: "INSERT",
        num: 0,
        measure: 4854
    },
    {
        label: "INSERT",
        num: 2,
        measure: 1254
    },
    {
        label: "CLEANUP",
        num: 0,
        measure: 105
    },
    {
        label: "INSERT",
        num: 3,
        measure: 45
    },
    ...
]
```

MongoDB/Client interface

Now that your MongoDB DB is filled with bunch of your measurements, you need to make the client understand what is your scheme. (See Supported Scheme to know if you need to follow this section)

For that, you will need to create a new *convertToSerie* adapter.

Create your own conversion adapter

Your *convertToSerie* adapter will transform your DB scheme into an array of values understandable by Highcharts library.

So, in order to make your DB scheme work with the client you will need to:

- check if Highcharts library can display your type of data

- create a new *convertToSerie* adapter function to match Highcharts way of displaying the data
- add a switch case to the *convertToSerieByChartType* switch function to link the highchart type to your custom adapter
- add your label and series type into the *$scope.labelTypeMap* map.

See the following Boxplot Example to understand where and how to do it.

Example: boxplot support implementation

YCSB produce a single value measure which is the latency. To support boxplot, we need to handle a measure object a bit more complex which looks like the following:

```
measure : {
    open: 50.45,
    high: 50.93,
    low: 46.61,
    close: 47.24
}
```

As explain, we need to design a new *convertToSerie* adapter which process the storage DB data and make it understandable for Highcharts:

```
// stats.controller.js

/**
 * Convert stored raw values from YCSB to Highchart formatting for candlestick series
 *
 * @param rawValues YCSB raw DB values
 * @returns {*} Highchart formatted data
 */
function convertToCandlestickSerie(rawValues) {
    return rawValues.map(function (measureObj) {
        return [
            measureObj.num,
            measureObj.measure.open,
            measureObj.measure.high,
            measureObj.measure.low,
            measureObj.measure.close
        ]
    });
}
```

Then we need to tell our client to use this adapter when candlestick type is selected so we add a switch entry to the *convertToSerieByChartType* switch function:

```
// stats.controller.js

/**
 * Select the conversion function based on the series type
 *
 * NOTE : you can add you own convertToSerie functions to support any series type!
 *
 * @param seriesType
 * @param rawValues
 * @returns {*}
 */
function convertToSerieByChartType(seriesType, rawValues) {
    var serie;
    switch (seriesType) {
        case "line":
            serie = convertToLineSerie(rawValues);
            break;
        case "candlestick":
            serie = convertToCandlestickSerie(rawValues);
            break;
        default:
            throw "Series type not supported yet, see our documentation to know how to implement it.";
            break;
    }
    return serie;
}
```

Then, we need to declare our label as a "candlestick" type series:

```
// stats.controller.js

$scope.labelTypeMap = {
    "AAPL Stock Price" : "candlestick"
};
```

Now, we might want to disable the average series by setting the *$scope.showAverage* to false, otherwise the average function will make average of our first value which is the *open* values, this doesn't make any sense. You still could modify the average functions if you really want the average series.

Finally, we want to support very large dataset so we added a new aggregating function within the switch of the aggregating route, *see app/routes/benchmarks.js for more details*.

Supported Scheme

The application supports only two schemes for the moment which are:

- The simple line scheme with *Highchart* "line" type

```
{
        label: "mylabel",
        num: 0,
        measure: 105
},
...
```

- The boxplot scheme with *Highchart* "candlestick" type

```
{
    label: "mylabel",
    num: 0,
    measure: {
        open: 50.45,
        high: 50.93,
        low: 46.61,
        close: 47.24
    }
}
```

Limitations

Linear loss of accuracy

As we see in MongoDB Aggregation Explaination Section, our application uses an aggregating process to handle millions of values. This aggregation process is reducing the precision of our displayed charts.

The precision reduction grow linearly when your benchmark points are increasing. The coefficient of this linear reduction is the value the user sets based on his computer performances. We have the following equation:

$$benchmarkLength = fixedUserConstant * bucketSize \Leftrightarrow bucketSize = \frac{benchmarkLength}{fixedUserConstant}$$

Bucket size precision equation

Almost stuck with MongoDB

You could rebuild the entire NodeJS MongoDB API to make it work with another DB but it wasn't our goal to achieve this compatibility.