Database Benchmark To Store And Retrieve Time Series Data in Raspberry Pi 3

I. Goal

Looking for most suitable database to store time series data in Raspberry Pi 3 that has fastest response time when called through Web API

II. Constraint

- Hardware and OS constraint : Using Raspberry Pi 3 with Raspbian OS 32 bit
- Data format constraint : Data stored as Json File
- Software constraint : There is available driver for .Net (if necessary)

III. Hardware and Software

Table 1

Hardware / Software	Specs / Version
Raspberry Pi 3 Model B Rev 1.2	CPU : ARMv7 rev 4 (600-1200 MHz) RAM : 924 MB OS : Raspbian GNU/Linux 10
DELL Vostro 14 3000 (local machine)	CPU : Intel Core i3-1005G1 (1.2 GHz) RAM : 8 GB OS : Windows 10
.Net core	5.0.102
RavenDB	5.1.5
CouchDB	3.1.1
Postman	-

IV. Result and Analysis

Source code can be accessed here : https://github.com/wandayusufalvian/rpi-nosql-webapi

4.1. DB Candidates :

🟩	promising
🟥	not promising

Table 2

No	Database	Information
🟥 1	MongoDB	https://stackoverflow.com/questions/51811140/how-to-install-mongodb-on-windows-7-32-bit "Changed in version 3.4: MongoDB no longer supports 32-bit x86 platforms." .Net driver for mongoDB is not support older version of MongoDB
🟩 2	RavenDB	>>install = https://www.youtube.com/watch?v=x4w_HfLWbX8 >>download = https://ravendb.net/download >>driver in .net = Raven.Client
🟩 3	CouchDB	>>install = https://github.com/jguillod/couchdb-on-raspberry-pi >>download=https://downloads.apache.org/couchdb/source/3.1.1/apache-couchdb-3.1.1.tar.gz >>driver in dotnet = mycouch. Alternative : use HTTP class in c# to communicate with database.
🟥 4	RethinkDB	https://github.com/rethinkdb/rethinkdb/issues/6449 "RethinkDB is impossible to use on a Raspberry Pi 3"
🟥 5	CockroachDB	https://github.com/cockroachdb/cockroach/issues/32147#issuecomment-438055230 "We don't currently support 32-bit architectures"
🟥 6	ArangoDB	so far, there is no clear evidence this DB can be installed in rpi 3 with Raspbian OS 32 bit
🟥 7	Cassandra	so far, there is no clear evidence this DB can be installed in rpi 3 with Raspbian OS 32 bit
🟥 8	OrientDB	so far, there is no clear evidence this DB can be installed in rpi 3 with Raspbian OS 32 bit
🟩 9	Json Plain Text	-

4.2. Create Dummy Time Series Data

Create dummy time series data for 1 day where each data consisted of datetime in each second and array of sensor data (5 sensors). Therefore there will be 86400 data. There are 2 kinds of data : Data 1 where all data will be stored as 1 array and Data 2 where each data will be delimited by a new line. Here are snippet of Data 1 and Data 2 :

Data 1

```
[{"datetime":"2021-01-01T00:00:00","sensorData":[0.6880378167554912,0.6316485664023312,0.5227921151196547,0.3506810224385378,0.8887706337910009]},
{"datetime":"2021-01-01T00:00:01","sensorData":[0.016884514138514415,0.5369466783185242,0.5169893244826185,0.3779079575920049,0.6729848276232299]}]
```

Data 2

```
{"datetime":"2021-01-01T00:00:00","sensorData":[0.5382447156767569,0.4929246285431667,0.20122024193462926,0.17451639248734171,0.7470556622124537]}
{"datetime":"2021-01-01T00:00:01","sensorData":[0.2841257542763491,0.8335463026694704,0.5241446627882052,0.282655261588588,0.6521198123936168]}
```

4.3. Experiment 1 :

- Using Data 1
- Install RavenDB and CouchDB in rpi 3
- Install .net core in rpi 3
- Create new ASP.NET Web API
- Install driver for RavenDB and CouchDB
- Install Newtonsoft.json
- Build and run Server and API in rpi 3
- Run server and database in rpi 3
- Use postman to call API and calculate response time
- Rpi 3 and local machine is connected through same network but indirectly (WIFI)

Table 3

No	Database	Data	Step	Response Time	Information
1	Json Plain Text	86400	1 s	25-28 s	Deserialize. Return IEnumerable
		86400	1 s	4-5 s	No Json Deserialize. StreamReader -> return string
2	RavenDB	86400	1 s	90-120 s	Hypothesis : it takes too long due to hardware. See table 4 for comparison

		8640	1 s	9-14 s	Hypothesis : it takes too long due to hardware. See table 4 for comparison
3	CouchDB	86400	1 s	16-17 s	-
		86400	5 s	3-4 s	-
		86400	10 s	1.5-1.7 s	-
		8640	1 s	1.6 s	-

>> Note on Json Plain Text

- Querying partial (with step) data will not affect the performance because all data will have to be deserialized or read as string. This is because in this experiment, the data that was used is Data 1 (all data as 1 array).
- It is faster to return a json file as string rather than deserialize it first. But, make sure the string is using the correct json format.

>> Note on RavenDB

- RavenDB performance is so disappointing when the server is built in rpi. For a comparison, Table 4 shows performance of RavenDB when the server is built in local machine :

Table 4

No	Data	Step	Response Time
1	86400	1 s	4-5 s
2	86400	5 s	0.6 - 1 s
3	86400	10 s	0.3 - 0.5 s
4	8640	1 s	0.3 - 0.5 s

- For results of step querying in Table 3 RavenDB can be regressed : From table 4, we can see that : (86400 data,10 s step) has the same response time compared to (8640 data,1 s step). From Table 3, we can see that : (86400 data,1 s step) have 10 x longer response time compared to (8640 data,10 s step). From these 2 informations, it can be inferred for Table 3 RavenDB that (86400 data, 5 s step) approximately : 18-24 s and (86400 data, 10 s step) approximately : 9-12 s.

>>Note on CouchDB

- Doesn't need to build ASP.Net Web API because it can be queried directly using HTTP. Example query to get all data : http://10.100.3.54:5984/test/_all_docs
- For more advanced query, unlike other DB where you write query anytime you need, in CouchDB you need to make your query using view first and save it.

>>Conclusion So Far :

- Json Plain Text is the best option so far when we just use simple query => retrieve all data.
- For the next experiment will be focused on using json plain text.

4.4. Experiment 2 :

The difference with Experiment 1 : Using ethernet cable directly from rpi 3 to local machine and only using json plain text.

Table 5

No	Response Time	Information

1	1.5-2.5 s	`using (StreamReader r = new StreamReader(jsonfile path)`
		`{`
		`      string json = r.ReadToEnd();`
		
		`return json;`
		`}`

2	1.6-2.5 s	`return File.ReadAllText(jsonfile);`

3	1.7-2.4 s	`using (StreamReader sr = File.OpenText(jsonfile path))`
		`{`
		`      StringBuilder sb = new StringBuilder();`
		`      sb.Append(sr.ReadToEnd());`
		`      return sb.ToString();`
		`}`

4	1.5-1.7 s	`using (FileStream fs = File.OpenRead(jsonfile path))`
		`using (BufferedStream bs = new BufferedStream(fs))`
		`using (StreamReader sr = new StreamReader(bs))`
		`{`
		`      string s=sr.ReadToEnd();`
		`      return s;`
		`}`

5	2-2.5 s	`using (StreamReader sr = File.OpenText(jsonfile path))`
		`{`
		`      StringBuilder sb = new StringBuilder();`
		`      String s = "";`
		`      while ((s = sr.ReadLine()) != null)`
		`      {`
		`            sb.Append(s);`
		`      }`
		`      return sb.ToString();`
		`}`

6	1.1-1.5	`//When this API is called, the json file is downloaded. there is no json file rendered in browser`
		`public IActionResult DownloadJson()`
		`{`
		`        string path = Directory.GetCurrentDirectory();`
		`        var net = new System.Net.WebClient();`
		`        var data = net.DownloadData(jsonfile path);`
		`        var content = new System.IO.MemoryStream(data);`
		`        var contentType = "application/json";`
		`        var fileName = "json-86400.json";`
		`        return File(content, contentType, fileName);`
		`}`

>>Notes on Suggested method in [Asynchronous PushStreamContent (stephencleary.com)](stephencleary.com) :

```
private static HttpClient Client { get; } = new HttpClient();
public async Task<HttpResponseMessage> Get()
{
    var stream = await
Client.GetStreamAsync("https://raw.githubusercontent.com/StephenClearyExamples/Async
DynamicZip/master/README.md");

    var result = new HttpResponseMessage(HttpStatusCode.OK)
    {
        Content = new StreamContent(stream),
    };
    result.Content.Headers.ContentType = new MediaTypeHeaderValue("text/plain");
    result.Content.Headers.ContentDisposition = new
ContentDispositionHeaderValue("attachment") { FileName = "README.md" };
    return result;
}
```

- This method's purpose is : download file (README.md) from a server (not our own server) indirectly through WebAPI without residing in our memory first, but there is a possibility that it resides first in its own server memory.
- Because it downloads the file, the browser does not render the file.
- Method 6 in Table 5 is quite the same with this suggested method.The difference is it downloads files from our own server.
- Method 6 in Table 5 is the fastest, but it directly downloads the file to harddisk. There is no returned json file that can be directly used by UI.

>>Conclusion so far : no significant differences using different methods, except for method 6 where it directly downloads files.

4.5 Experiment 3

- Using Data 2, therefore it can select data with steps.
- Using method 1 and 4 because both are the fastest compare to others (except for method 6 which directly download the file)

Table 6

No	Step	Response Time
1	1 s	2-3 s
2	2 s	1.4-1.6 s
3	5 s	0.93-0.97 s
4	7 s	0.85 -0.9 s
5	10 s	0.77-0.82

>>Notes on Table 6 :

- Result for 1 s step in Table 6 is different compared to Table 5 because there is an additional step => add selected stream to stringbuilder.

```
public static string ReadJsonEachLines2(int step)
{
            StringBuilder sb = new StringBuilder();
            string path = Directory.GetCurrentDirectory();
            string rpiPath = @$"{path}/Data/json-86400.json";
            using (StreamReader r = new StreamReader(rpiPath))
            {
                int i = 0;
                while (i < 86400)
                {
                    if (i % step == 0)
                    {
                        sb.Append(r.ReadLine() + Environment.NewLine);
                    }
                    else { r.ReadLine(); }
                    i++;
                }
            }
            return sb.ToString();
}
```

- Add steps (2 s, 5 s, 7 s, 10 s) reduce response time compared to returning all data in Table 5.
-

IV. Conclusion and Suggestion

- Json plain text is most suitable compare to RavenDB and CouchDB because it has fastest response time
- Add steps reduce response time
- Suggestion : use json plain text with 5 s steps to get enough data to represent all data and get quite fast response time (under 1 second)