DATA White Paper

The content of this Guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Evarist Fangnikoue; Evarist Fangnikoue assumes no responsibility or liability for any errors or inaccuracies that may appear in this publication. Evarist Fangnikoue, reserves the right to change information in this publication without notice, as a result of

System enhancements or other reasons. The works of authorship contained in this publication, including but not limited to all design, text and images and the software described herein, are owned, except as otherwise expressly stated, by Evarist Fangnikoue , or its affiliates or licensors. The entire contents of this publication are protected by Hong Kong and worldwide copyright laws and treaty provisions. In accordance with these terms, except as stated above, you may not copy, reproduce, modify, use, republish, upload, post, transmit or distribute in any way material from the publication. Further, you may not copy, modify or display any of Evarist Fangnikoue's work appearing in this publication in any way without Evarist Fangnikoue's express written consent. Evarist Fangnikoue assumes no responsibility or liability for any errors or inaccuracies that may appear in this publication. Except as permitted by such license, no part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, or otherwise, without the prior written permission of Evarist Fangnikoue.

RESTRICTED RIGHTS LEGEND

This publication is provided with "Restricted Rights". No part of this publication may be photocopied, reproduced or transmitted, in any form or by any means, without the prior written consent of Evarist Fangnikoue.

EXPORT REQUIREMENTS

This item is subject to Hong Kong export control laws and regulations. This item may not be exported, re-exported, re-transferred, disclosed or otherwise diverted contrary to Hong Kong export control laws and regulations.

NO WARRANTY

THE CONTENTS OF THIS PUBLICATION ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF QUALITY, MERCHANTABILITY,FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

Evarist Fangnikoue SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED AS A RESULT OF USING THE CONTENTS OF THIS PUBLICATION. IN NO EVENT SHALL Evarist Fangnikoue BE LIABLE FOR ANY INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL, OR

CONSEQUENTIAL DAMAGE (INCLUDING LOSS OF BUSINESS, REVENUE, PROFITS, USE, DATA OR OTHER ECONOMIC ADVANTAGE) HOWEVER IT ARISES, WHETHER FOR BREACH OR IN TORT, EVEN IF Evarist Fangnikoue HAS BEEN PREVIOUSLY

ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. BECAUSE SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

IT IS IN ALL CASES THE USER'S RESPONSIBILITY TO ENSURE THAT THE PRODUCTS ARE PROGRAMMED AND USED IN ACCORDANCE WITH ALL APPLICABLE LAWS AND REGULATIONS AND IN A MANNER.

Contents

1. Authorization page

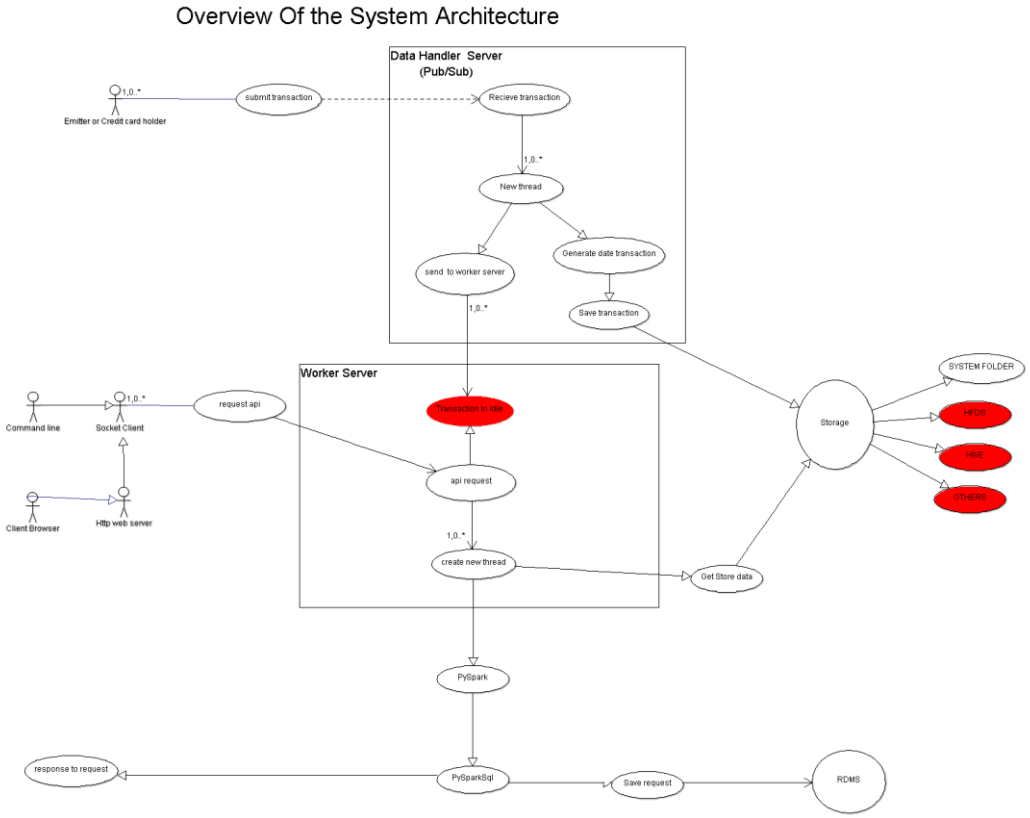
Preparation: This document is prepared by the following:			
Prepare By	**Version**	**Signature**	**Date**
Fangnikoue K. Ayao	0.1	F.K.A.	August 2018

2. Overview

In the problem looking for a solution to handle millions of transaction data a day from credit card holder, this white paper will describe some procedure that can be use other to solve the problem, the architecture including the work describe in this paper , is only in the earlier version of solving the problem describe above. The collect of those data will help the company to predict the customer behavior using the credit card for days to come. With time and in the light of more research, this white paper can offer a sophisticate algorithm to solve the problem in efficient way.

3. System Architecture and Functionality

3.1 Use Case



Double click to view

UseCaseDiagram2.svg

3.2 Subordinate Use Case

3.2.1 Receive Transaction

1- Create Transaction Thread handler
2- Storage Transaction
3- Send Transaction to worker server

3.2.2 Request API

1 Create Api request Thread
2 Get Store data
3 Send to PySpark

4. Description of the Solution Implemented

The Solution provided in this white paper can be extends further, as in the current implementation , I am running 2 separate servers in this eco-system such the Data server and the Worker server respectively to handle the send transaction and to proceed an api requested by x user,. The use case diagram attached to this paper can give you an overview of the architecture implemented.

1- Data Handler Server(Pub/Sub)
The Data Handler server is an asynchronous process toward any client request. The main function of this server, is to wait for the client or emitter request in json format. Once the request has been receive, it disconnect the requester and proceed with the data manipulation, to do so, the handler server will create a new thread to proceed this request, the thread will first send the receive data to the Worker server for real time processing, but this implementation has not been implement here but can be in near future, then generate a temporary transaction file with the date and hour that the transaction has been receive. The Process of this storage is base per hour, all the transaction that has been submit in that current hour will be saved in the same temporary file. The process of data storage is handle in a way that if the file exists, the Data server will just append the new data into it, otherwise, it (The data handler server) will create a new temporary file.
The format of the file representation is: “yyyy_mm_dd_hh.json” .The sample name of this file is: “2018_08_18_10.json” which represent all the transactions proceed on 18 of August between 10:00 and 10:59 and the circle continue.

2- Worker Server:
The worker server is just an API server that get the user request and proceed it. Remember that this server is not retrieving the data from the database but rather from the disk using PySpark. The process of this approach is handle in

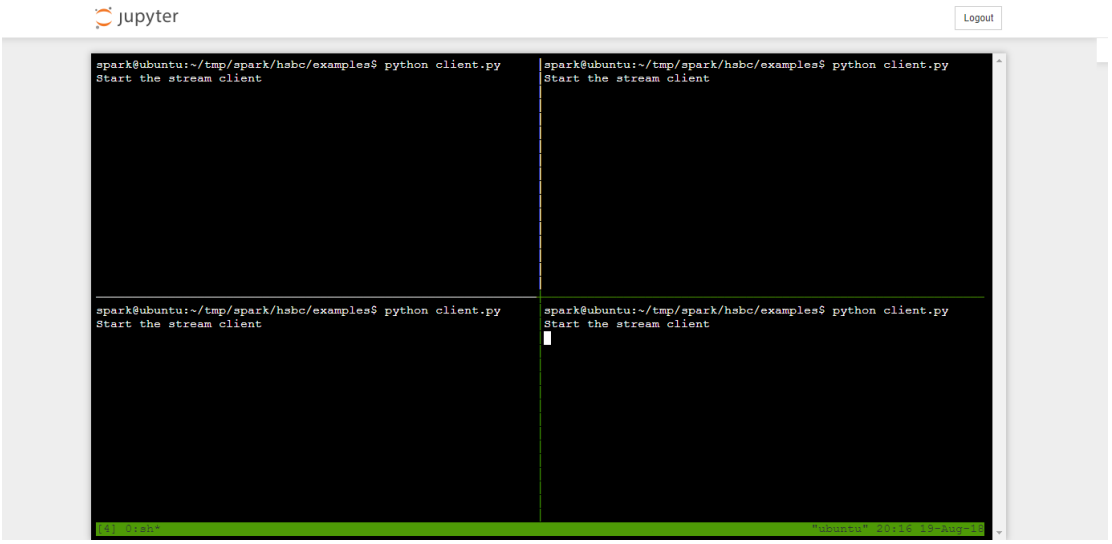
the way that whenever, there is a new API request, the worker server will create a new thread to handle that request, assume, the api requester want to view all the transactions that has been happen now, the worker sever thread will go to the location where the temporary file has been store and get the file that matches the date and hour. Example: Assume that the requester is living in Hong Kong and his time is August 21 in the year 2018 and his current time is 11:04:34, the worker server will translate the request as follow: 2018_08_21_11 and from that, it will just load a file name: "2018_08_21_11.json" that has all the transactions between 11:00 until 11:04, it looks like a real time processing but I do not think so.

Once the process has been complete, it will send the response to the user first, and then proceed to save the request in the database.

The request time can be adjust, for example you can request a time interval X with X>=0.

Demo Example:
Here I am using jupyter browser to perform the test case.
There are 4 socket clients running at the same time as emitters. Each of the emitter is simulate 4 credit card customers at the same time with the same name, same credit card number and different transaction amount, different time and different recipient bank account number. Each cycle take 2 seconds, so in this test case with 4 emitters, there is a total of 16 transactions in 2 seconds, 480 transactions in a minute and 28800 transaction an hour. But the emitters time delay can be adjust.

The figure below show the running of the 4 emitters.

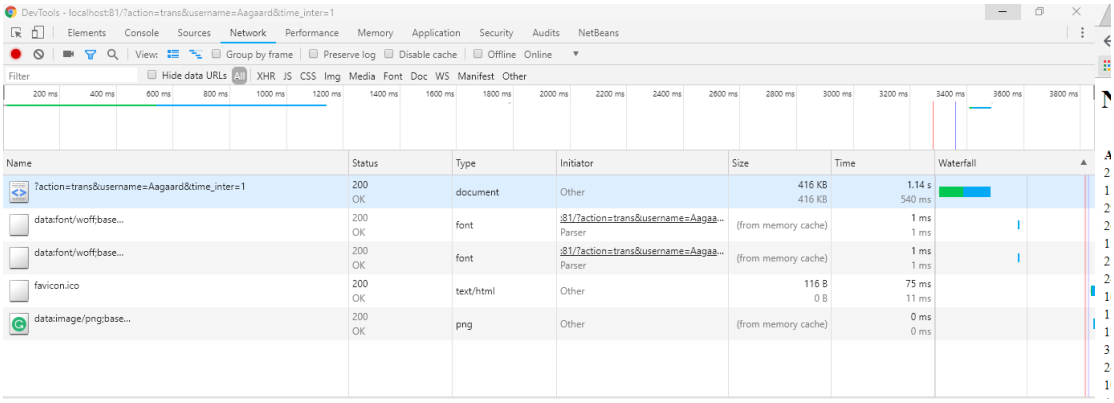


The figure below just show some test case, that I had perform using the browser, the current implementation support both browser request and command line request, for the browser request , I only implement the Get request. Further in the below figure we are seeing a user name Aagaard has perform 2275 transaction in 2 hours as showing in the figure.

Number of record found:2275

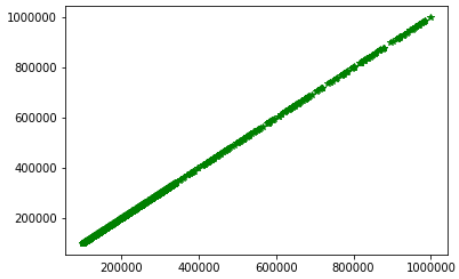
Amount	Date Time	Transaction Recipient Card Primary Account Number	Sender Account Number	Sender Country Code	Sender Name	Source Of Funds	Note	Transaction Currency	Action
259886	2018-08-19 21:00:01	102139723628791016833 2837	225983865630473673903 9602	USA	Aagaard	None		USD	trans
175959	2018-08-19 21:00:02	246039993288040624685 2079	316576574219867801563 7610	USA	Aagaard	None		USD	trans
292542	2018-08-19 21:00:04	200442921039989019098 0686	239926527233602841302 3663	USA	Aagaard	None		USD	trans
268424	2018-08-19 21:00:11	335333852843739500113 5066	221290845266015873073 2537	USA	Aagaard	None		USD	trans
181262	2018-08-19 21:00:13	570137538436093176011 1831	225983865630473673903 9602	USA	Aagaard	None		USD	trans
258806	2018-08-19 21:00:14	285466785214688061785 5599	316576574219867801563 7610	USA	Aagaard	None		USD	trans
242430	2018-08-19 21:00:16	318338969589133731432 8189	239926527233602841302 3663	USA	Aagaard	None		USD	trans
169031	2018-08-19 21:00:23	431070977532819805495 3353	221290845266015873073 2537	USA	Aagaard	None		USD	trans
135408	2018-08-19 21:00:25	850134594234027511528 1921	225983865630473673903 9602	USA	Aagaard	None		USD	trans
121012	2018-08-19 21:00:27	740529152219056112919 2066	316576574219867801563 7610	USA	Aagaard	None		USD	trans
310821	2018-08-19 21:00:29	114155481600855203219 7920	239926527233602841302 3663	USA	Aagaard	None		USD	trans
245357	2018-08-19 21:00:35	117390017394028789142 3352	221290845266015873073 2537	USA	Aagaard	None		USD	trans
101076	2018-08-19 21:00:37	331438169433911700533 4973	225983865630473673903 9602	USA	Aagaard	None		USD	trans
450343	2018-08-19 21:00:39	315324234800821548719 9143	316576574219867801563 7610	USA	Aagaard	None		USD	trans
196812	2018-08-19 21:00:41	298444073275154036526 3963	239926527233602841302 3663	USA	Aagaard	None		USD	trans
325284	2018-08-19 21:00:47	194223551685167237224 5149	221290845266015873073 2537	USA	Aagaard	None		USD	trans
111100	2018-08-19 21:00:49	909957010303301597676 905	225983865630473673903 9602	USA	Aagaard	None		USD	trans
112445	2018-08-19 21:00:51	338566731878548339247 9990	316576574219867801563 7610	USA	Aagaard	None		USD	trans
709675	2018-08-19 21:00:53	170325942829290224005 1586	239926527233602841302 3663	USA	Aagaard	None		USD	trans
154319	2018-08-19 21:00:59	212192601254332732068 6133	221290845266015873073 2537	USA	Aagaard	None		USD	trans
682812	2018-08-19 21:01:01	688660369006720002451 0956	225983865630473673903 9602	USA	Aagaard	None		USD	trans
172540	2018-08-19 21:01:03	224509740568888181298 7978	316576574219867801563 7610	USA	Aagaard	None		USD	trans
732213	2018-08-19 21:01:05	255842649715742110546 1529	239926527233602841302 3663	USA	Aagaard	None		USD	trans
310636	2018-08-19 21:01:11	289203572581874889763 6902	221290845266015873073 2537	USA	Aagaard	None		USD	trans
112354	2018-08-19 21:01:13	304365046045596036935 471	225983865630473673903 9602	USA	Aagaard	None		USD	trans
414390	2018-08-19 21:01:15	199443883110221282719 5671	316576574219867801563 7610	USA	Aagaard	None		USD	trans
279172	2018-08-19 21:01:17	224002751944348745311 6140	239926527233602841302 3663	USA	Aagaard	None		USD	trans
262011	2018-08-19 21:01:23	299507070391426899314 397	221290845266015873073 2537	USA	Aagaard	None		USD	trans
376317	2018-08-19 21:01:25	411975329405422360630 5777	225983865630473673903 9602	USA	Aagaard	None		USD	trans

As the figure below showing the time process of the request, the http server take less than 1.14 second to proceed a retrieve a total record of 2275. But using a command line request it take less than one second to respond to the request.



The below show a simple sum of transaction diagram print using jupyter notebook.

```
+-----------+
|sum(Amount)|
+-----------+
|358354301  |
+-----------+
```



5. API Request Reply (APIRR)

Restful API List		
End Point	Request Header	Request Body
action	GET - Send user and time line information.	User information including the time line interval for the transaction to view.

5.1 Action GET value

Restful API Request Value			
End Point	Data type	Description	Return Value
trans	String	Display all the list of transaction by the given user and time interval.	Browser: Html format
			CLI: List or Json
sum	String	Give the sum of all the transaction by the requested user name and time line	Browser:Html
			CLI: List or json

5.2 Request Body

Object: action		
Field name	Description	Format
username	This parameter represent the username in which the requester would like to view the information	String
time_inter	The time interval for the transaction.	String or integer For the String data type it only accept “Now” or “now”

5.3 Action

5.3.3 Transaction GET request using the browser

Function : view list of transaction
Usage : GET http://system_ip/?action=trans&username=Aagaard&time_inter=1 Request body : No Response body: Html data
Function: sum of the transaction
Usage :

GET http://system_ip/?action=sum&username=Aagaard&time_inter=1 Request body : No Response body: Html data

5.4 Error Handler

5.4.1 Return Error Message

Error message	Description
`No transaction found`	There is no record found to the given user or time interval requested

5.4.2 Return Status

Status	Description
success	Indicated that the request is successful.
failed	Indicated that the process has failed see the “message” for more information.

6. Implementation time line

Assignment Task List

Tasks	Forecast Start	Forecast End	Man Days	Man Hours	Status
First uncounter with PySpark and Learning	8/14	8/15	1	0	Completed
Understand the assignment and further reseach	8/15	8/15	0	2	Completed
1- Install a VM for the assignment environment 2- Download and Configure Spark 3- Download python Library 4-Other configuration	8/15	8/15	0	7	Completed
Creating the emitter	8/16	8/16	0	3	Completed
Creating the Data handler Server	8/16	8/16	0	3	Completed
Creating the worker server and simple http web server for testing	8/16	8/17	0	7	Completed
Creating a command line Api requester	8/17	8/17	0	2	Completed
Preparaing the assignment documentation and manual	8/20	8/20	0	8	Completed
Total			5	0	

7. Conclusion

A lot of research can be applied in a way that this implementation can solve the problem describe above in an efficient way. Imagine this implementation will be move into a real system, I would suggest to further, investigate on running multiple cluster of the Data handler and Worker server, implement a load balancer in a way that when one server is down the other will be active. As I describe above working ,with the file in this implementation is still look us a real time implementation, but in the use case diagram of the system architecture, the real-time implementation use case is mark in red colour which mean that is not implemented in this stage. But can be implement. And for others storage other than the system folder storage used in this implementation, it will be good to test them and check the performance against each other and do more research to further compare the result. In this test case there are 28800 transactions per hour but imagine there are millions, how will the system will respond, this answer will be in linear space for further investigation.

The above implementation use the credit card transaction data, but the architecture can remain the same to work with other data source.

The second version of this white paper will demonstrate, the use of machine learning to check the request data and allow the system to make his own decision according to the data receive. Further to the implementation describe above, it will be straight

forward, to encrypt the requested data and share that data among a listed of connected nodes that will run continuously without a down time. Each of those nodes will carry the requested data, which will be duplicated across the node eco-system.