

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	4
2. Overview	4
3. System Architecture and Functionality	5
3.1 Use Case	5
3.2 Subordinate Use Case	5
3.2.1 Receive Transaction	5
3.2.2 Request API	6
4. Description of the Solution Implemented	6
5. API Request Reply (APIRR)	10
5.1 Action GET value	10
5.2 Request Body	10
5.3 Action	11
5.3.3 Transaction GET request using the browser	11
5.4 Error Handler	12
5.4.1 Return Error Message	12
5.4.2 Return Status	12
6. Implementation time line	Error! Bookmark not defined.
7. Conclusion	13

Abstract:

Today Terabyte of informations been store in your company or millions of company customers data has been receive every seconds or minutes. Processing thoses data can be painfull. In a traditional way , company collect thoses data, save it in the database and use sql querry to process it. Not only the data has been centralise in one location but can also easily be destruct when there is a security breach in your network environemnt. The processing time for real time decision making can be problematic using the above approach. In this work we present a potential of data streamming technology using Python along side with the PySpark technology without a need of a centralize system for data processing.

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

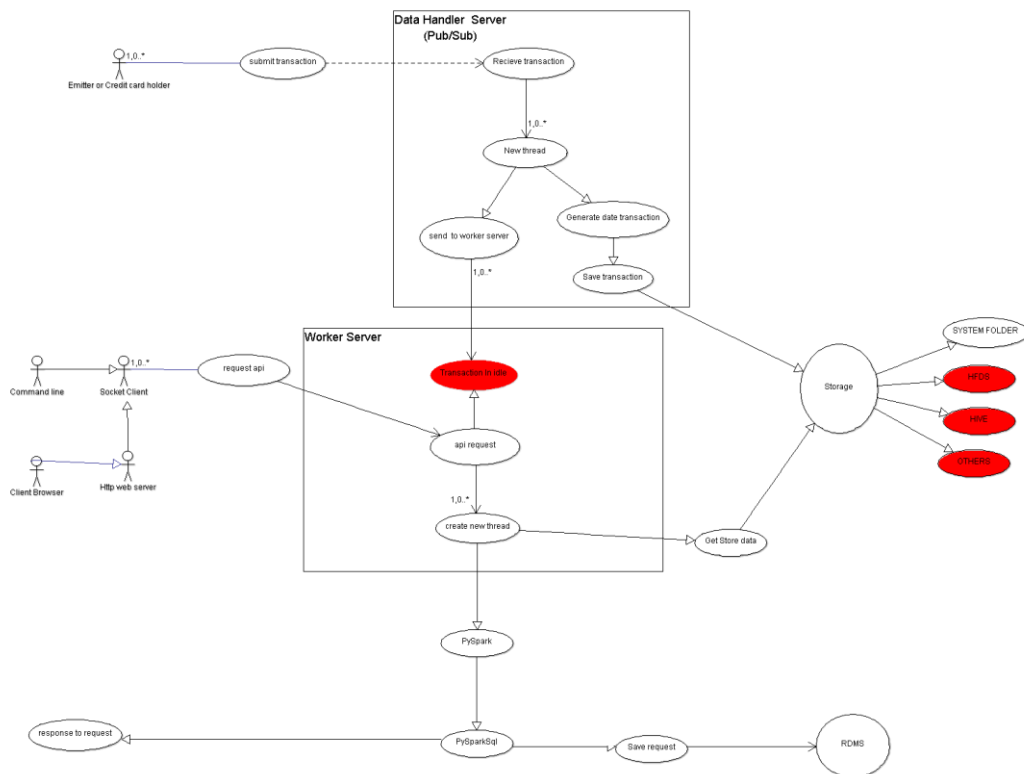
2. Overview

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

Overview Of the System Architecture



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.

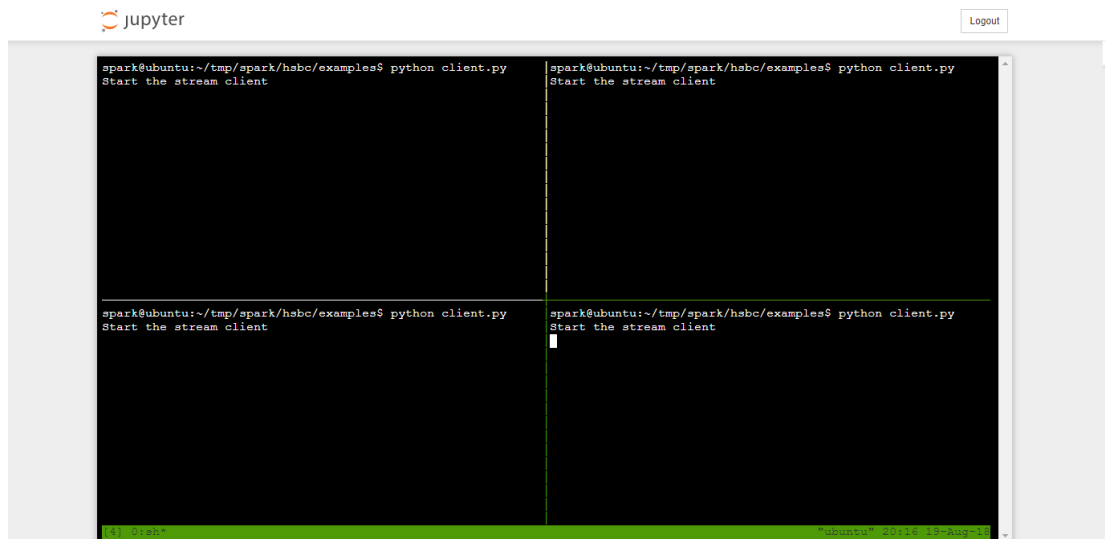
The request time can be adjust, for example you can request a time interval X with $X \geq 0$.

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.



```
spark@ubuntu:~/tmp/spark/habc/examples$ python client.py
Start the stream client

spark@ubuntu:~/tmp/spark/habc/examples$ python client.py
Start the stream client

spark@ubuntu:~/tmp/spark/habc/examples$ python client.py
Start the stream client

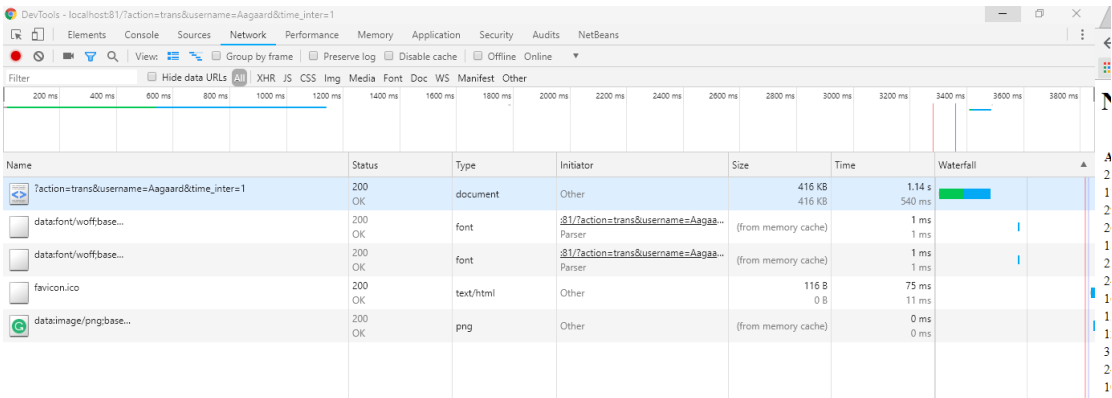
spark@ubuntu:~/tmp/spark/habc/examples$ python client.py
Start the stream client
```

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	1021397236287910168332837			2259838656304736739039602	USA		Aagaard	None		USD		trans
175959	2018-08-19	21:00:02	2460399932880406246852079			3165765742198678015637610	USA		Aagaard	None		USD		trans
292542	2018-08-19	21:00:04	2004429210399890190980686			2399265272336028413023663	USA		Aagaard	None		USD		trans
268424	2018-08-19	21:00:11	3353338528437395001135066			2212908452660158730732537	USA		Aagaard	None		USD		trans
181262	2018-08-19	21:00:13	5701375384360931760111831			2259838656304736739039602	USA		Aagaard	None		USD		trans
258806	2018-08-19	21:00:14	2854667852146880617855599			3165765742198678015637610	USA		Aagaard	None		USD		trans
242430	2018-08-19	21:00:16	3183389695891337314328189			2399265272336028413023663	USA		Aagaard	None		USD		trans
169031	2018-08-19	21:00:23	4310709775328198054945353			2212908452660158730732537	USA		Aagaard	None		USD		trans
135408	2018-08-19	21:00:25	8501345942340275115281921			2259838656304736739039602	USA		Aagaard	None		USD		trans
121012	2018-08-19	21:00:27	7405291522190561129192066			3165765742198678015637610	USA		Aagaard	None		USD		trans
310821	2018-08-19	21:00:29	1141554816008552032197920			2399265272336028413023663	USA		Aagaard	None		USD		trans
245357	2018-08-19	21:00:35	1173900173940287891423352			2212908452660158730732537	USA		Aagaard	None		USD		trans
101076	2018-08-19	21:00:37	3314381694339117005334973			2259838656304736739039602	USA		Aagaard	None		USD		trans
450343	2018-08-19	21:00:39	3153242348008215487199143			3165765742198678015637610	USA		Aagaard	None		USD		trans
196812	2018-08-19	21:00:41	2984440732751540365263963			2399265272336028413023663	USA		Aagaard	None		USD		trans
325284	2018-08-19	21:00:47	1942235516851672372245149			2212908452660158730732537	USA		Aagaard	None		USD		trans
111100	2018-08-19	21:00:49	909957010303301597676905			2259838656304736739039602	USA		Aagaard	None		USD		trans
112445	2018-08-19	21:00:51	3385667318785483392479990			3165765742198678015637610	USA		Aagaard	None		USD		trans
709675	2018-08-19	21:00:53	1703259428292902240051586			2399265272336028413023663	USA		Aagaard	None		USD		trans
154319	2018-08-19	21:00:59	2121926012543327320686133			2212908452660158730732537	USA		Aagaard	None		USD		trans
682812	2018-08-19	21:01:01	6886603690067200024510956			2259838656304736739039602	USA		Aagaard	None		USD		trans
172540	2018-08-19	21:01:03	2245097405688881812987978			3165765742198678015637610	USA		Aagaard	None		USD		trans
732213	2018-08-19	21:01:05	2558426497157421105461529			2399265272336028413023663	USA		Aagaard	None		USD		trans
310636	2018-08-19	21:01:11	2892035725818748897636902			2212908452660158730732537	USA		Aagaard	None		USD		trans
112354	2018-08-19	21:01:13	3043650464045596036935471			2259838656304736739039602	USA		Aagaard	None		USD		trans
143490	2018-08-19	21:01:15	1994438831102212827195671			3165765742198678015637610	USA		Aagaard	None		USD		trans
279172	2018-08-19	21:01:17	2240027519443487453116140			2399265272336028413023663	USA		Aagaard	None		USD		trans
262011	2018-08-19	21:01:23	2995070703914268899314397			2212908452660158730732537	USA		Aagaard	None		USD		trans
376317	2018-08-19	21:01:25	4119753294054223606305777			2259838656304736739039602	USA		Aagaard	None		USD		trans
238851	2018-08-19	21:01:27	200737014737370761485687			3165765742198678015637610	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.



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. 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.