# Data Driven Debugging

Daigo Tanaka

February 1, 2015

```
Teardown UDP connection for faddr 194.114.201.13/48355 gaddr 10.0.0.
Mar 29 2004 09:56:35: %PIX-6-302006:
        2004
                                             Built UDP connection for faddr 194.114.201.13/48355 gaddr 10.0.0.187 Teardown UDP connection for faddr 194.114.201.13/48355 gaddr 10.0.0. Built UDP connection for faddr 194.114.201.13/48355 gaddr 10.0.0.187
               09:56:35:
                           %PIX-6-302005:
                           %PTX-6-302006:
Mar
               09:56:35:
        2004
Mar
               09:56:35:
                           %PIX-6-302005:
        2004
               09:56:35:
                           %PIX-6-302006:
                                              Teardown UDP
                                                             connection for faddr 192.168.216.1/137
    29
29
                                             Teardown UDP connection for faddr 203.124.140.107/12520 gaddr 10.0.0
Mar
        2004 09:56:35:
                           %PIX-6-302006:
        2004
Mar
              09
                       15:
                           %PIX-6-302006:
                                             Teardown UDP
                                                             connection for faddr 195.146.160.3/16708 gaddr 10.0.0.1
                                                              connection 152017 faddr 212.56.240.37/9200 gaddr 10.0.0 connection for faddr 212.56.240.37/9200 gaddr 10.0.0
        2004
               09
                           %PIX-6-302006:
                                              Teardown UDP
Mar
     29
        2004
                           %PIX-6-302002:
                                              Teardown TCP
                                                             connection for faddr 194.25.0.125/38729 gaddr 10.0.0.18 connection for faddr 211.9.32.235/32770 gaddr 10.0.0.18 connection for faddr 194 114 201.14/46474 gaddr 10.0.0.
        2004
Mar
     29
               09
                       15:
                           %PIX-6-302006:
                                              Teardown UDP
               09
                           %PIX-6-302006:
                                              Teardown UDP
Mar
                       15:
Mar
        2004
               09
                           %PIX-6-302006:
                                              Teardown
        2004
                           %PIX-6-302001:
    29
        2004
              09
                       18:
Mar
                           %PIX-5-304001:
                                             192.1/
                                                                         RL 216.52.
                                                                                                            Lcom/0/G.1-XP-R/s1
                                                                                                            '80 gaddr 10.0.0.1
.52.17.116/80 gadd
                                                         CP conne.
                                                                                           216.52.
                       19:
                           %PTX-6-302002:
Mar
                                             Tear
                                                                            152070
Mar
        2004
                           %PIX-6-302001:
                                                       tbound TCP
                                                                                        /1 for fadd.
                                             Bui
                                                                            ction
        2004
                           %PIX-5-304001:
                                                       .0.2 Accessed
                                                                                        .116:/b/ss/n
                                                                                                             com/0/G.1-XP-R/s1
Mar
        2004
               09
                       9: %PIX-6-302002:
                                                      in TCP connecti
                                                                             5207
                                                                                       ddr 216.52.17.
2.168.202.1/13
2.168.216.1/13
                                                                                                             '80 gaddr 10.0.0.1
                                             Te:
                                                                                                            addr 10.0.0.187/27
               09
                       0: %PIX-6-302005:
                                                     JDP connection
Mar
                                             Bu
                                                                             fadc
        2004
                                                                                                            iddr 10.0.0.187/27
Mar
                           %PIX-6-302005:
                                             Bu
                                                     JDP
                                                          connection
                                                                             fadc
        2004
                           %PIX-4-106023:
                                                          src outside
                                                                             e-Ci
                                                                                       dst inside:10
                                                                                                             187
                                                                                                                  (type 3,
                                                                                                            ldr 10.0.0.187/53
.187 (type 3, code
l gaddr 10.0.0.187
    29
29
        2004
Mar
               09
                       2: %PIX-6-302005:
                                             Bu
                                                      IDP
                                                          connection
                                                                             fadd
                                                                                       58.4.34/3707
               09
                       2: %PIX-4-106023:
Mar
                                             Der
                                                          src outsid
                                                                            le-Cis
                                                                                        st inside:10
        2004
                           %PIX-6-302005:
                                                                             faddr
Mar
                                             Bui.
                                                                                         114.201.14
                                                          connection
     29
        2004
               09
                                                         'IDP conne
                                                                                                                  gaddr 10.0.0.
                                              Tear
                                                                            for fad
                                                                                                            l gaddr 10.0.0.187
        2004
Mar
     29
               09
                                              Built
                                                                         or faddr 1:
                                                                                                            .187 (type 3, 5474 gaddr 10
                                             Mar
               09
                                                                                                                             code
        2004
Mar
               09:56:53: %PIX-6-302006:
                                              Built UDP connection for
                                                                                                              gaddr 10.0.0.187
        2004
               09:56:53:
                           %PIX-6-302005:
                                                                                    194.114.201.14/
                                                                             faddr
                                                                                                         69 gaddr 10.0.0.18
/32769 gaddr 10.0.0
2769 gaddr 10.0.0.18
Mar 29
Mar 29
              09:56:53:
        2004
                           %PIX-6-302005:
                                             Built UDP connection for faddr
                                                                                        .120.214.16
        2004
                           %PIX-6-302006:
                                             Teardown UDP connection for fa
                                                                                         19.120.21
Mar
        2004
               09:56:53:
                           %PIX-6-302005:
                                              Built UDP connection for
                                                                             faddr
                                              Deny TCP (no connection)
                                                                                                        to 192.168.202.1/2244
     29
        2004
              09:56:55:
                           %PIX-6-106015:
                                                                             from 192.
                                                                             from 192.168.u.z/2799 to 192.168.202.1/2244
Mar
     29
        2004
               09:56:58:
                           *PTX-6-106015:
                                             Deny TCP (no connection)
                                             Built UDP connection for faddr 66.246.44.108/59213 gaddr 10.0.0.187/
Teardown UDP connection for faddr 66.246.44.108/59213 gaddr 10.0.0.1
     29
        2004
               09:57:02:
                           %PIX-6-302005:
Mar
        2004
               09:57:02:
                           %PIX-6-302006:
               09:57:02:
                           %PIX-6-302005:
                                              Built UDP connection for
                                                                             faddr 66.246.44.108/59213 gaddr 10.0.0.187/
Mar 29 2004 09:57:04:
                           %PIX-6-106015:
                                             Deny TCP (no connection) from 192.168.0.2/2799 to 192.168.202.1/2244
```

In this article, I share my recent experience in data-driven debugging. Data-driven debugging is the practice of using data to trouble-shoot software defects efficiently. It is not a new idea at all. In fact, most software engineers do it already. But I think we could practice it more consciously because it may work to cut the debugging of complex systems from days to a few hours.

## Case 1: Web socket communications

Consider a pair of devices at a store like Walgreens. One is facing the cashier and the other is facing the customer. As the cashier processes the purchase, relevant information is displayed on both devices in different presentations to aid the communication between the customer and the cashier.

Such a system can be built with two tablets and a web socket server as shown in Fig. 1. Because the cashier-side tablet (Tablet B) needs to process some kind of transaction, it sends Application Programming Interface (API) calls to the API server that reads and writes data to the database. The web socket server also uses the same API and database to retrieve information about the mapping of device pairs so it can handle many pairs in different stores.

<sup>&</sup>lt;sup>1</sup>In this architecture, the customer side tablet (Tablet A) does not initiate any transaction. This allows the entire workflow logic to exist in Tablet B for the consistency of the transactions. Tablet A stay as "dumb terminal" that only displays what it is told to display.

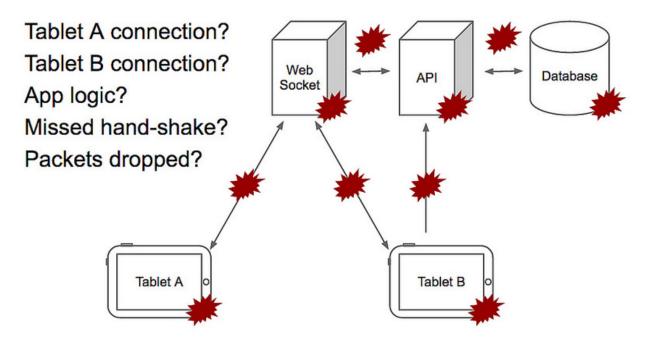


Figure 1: Paired tablets communicating via web socket server with potential points of failure highlighted by red "bangs".

Many points of failure, so a large hypothesis space. This is a fairly complex system. Each component from the tablet to the database is subject to a logical failure, and each line of communication is also a point of failure.

If we get a report from the customer support team that many tablets stopped talking to each other, what would we suspect?

- Was Tablet A not recovering the web socket connection after boot?
- Was Tablet B not recovering the web socket connection?
- Both tablets are connected to a web socket but is there an application logic failure preventing the correct information from being displayed?
- Did one of the pair miss the handshake signals from the other?
- Was the store Wi-fi signal not reliable? (the tablets are connected to the Internet via Wi-fi)

In a complex system, the number of failure scenarios (or hypotheses) dramatically increases (i.e. the number of nodes plus the number of edges connecting them). Investigating just one of the hypotheses could be a lot of work. It could easily take days to get to the bottom of the issue if we exhaustively investigate one hypothesis after another.

Reduce the hypothesis space based on the evidence. So, it is necessary to reduce the hypothesis space before jumping into the code-level investigation. Analyzing logs will help to achieve it.

Luckily, in our hypothetical case, we have a log from the web socket server. After extracting the portion of the logs relevant to the affected store and date, we may notice a pattern as shown in Fig. 2.

```
Jan 6 18:05:45: Opened connection for TABLET-A
                                                          1) Weird!
Jan 6 18:06:40: Opened connection for TABLET-B
                                                          Tablet A has two
Jan 6 18:06:40: Mapped TABLET-A and TABLET-B
                                                          open sockets?!
Jan 6 18:22:56: Opened connection for TABLET-A
Jan 6 18:22:56: Mapped TABLET-A to TABLET-B
Jan 6 18:22:57: Message from TABLET-B (action: "handshake"...
                                                                   2) Also weird.
Jan 6 18:22:57: Closed connection for TABLET-A
                                                                   Message from
Jan 6 18:22:57: Removed Mapping of TABLET-A and TABLET-B
                                                                   Tablet A after
Jan 6 18:22:58: Message from TABLET-A {action:"reset"...
                                                                   closing the
                                                                   socket.
```

Figure 2: Log analysis

At the beginning, there was an existing connection from both Tablets A and B to the web socket server at 18:05:45. A mapping between the tablets was created just after the connection opened at 18:06:40.

Then a strange thing happened. There was another open connection event by Tablet A. This is odd because there wasn't a "close connection" event log before this. It seems that there are two open sockets for Tablet A at 18:22:56.

What happened next was Tablet A closes the connection just after receiving a handshake message from Tablet B at 18:22:57. But strangely, Tablet A still sent a message at 18:22:58. So, Tablet A wasn't disconnected after all?

Based on the log analysis, and the knowledge of the software and store environment, one could hypothesize the following failure scenario:

Store Wi-fi was weak as often happens in the local retail store environment. Suddenly, the weak signal caused the socket of Tablet A to disconnect in an unclean way. So, the disconnection wasn't recognized by the web socket server immediately. Tablet A, regaining the Wi-fi signal, sends a socket open request to the web socket and it is accepted. When Tablet B sends a handshake signal, the disconnection of the previous (and lost) socket (I call this a zombie socket) was finally recognized by the web socket server. In this software, the mapping information between Tablets A and B is removed upon the closing of the socket. So, when the zombie socket was finally reaped, the server removed the mapping information even though another live socket still existed for Tablet A. As a result, the socket for both tablets existed, but the mapping between the two was broken.

**Lessons learned** This turned out to be exactly what was happening, and we could fix it quickly. Even if this had not been the case, we could rule out many scenarios that do not conform to the evidence from the log.

#### Lessons learned:

- 1. Try reducing the hypothesis space first based on evidence when debugging a complex system.
- 2. Make sure that the application logs at each critical step.
- 3. Send the client log to the server.

## Case 2: Discrepancy between two data sources

The second example is about data discrepancy between systems. Suppose we need to send the account information on the proprietary database (SQL) to Salesforce so that the sales department can use it for sales and customer retention efforts.

One day, we receive an email saying that the data in SQL and Salesforce have a discrepancy. We first need to investigate by comparing the data between the two sources.

**Analyzing data from multiple sources** This case is not so much about reducing the hypothesis space. Instead we needed to carefully compare the data between two systems: SQL and Salesforce.

What tool do we use to start analyzing? It may sound very strange to application programmers, but my choice was R. R has packages to connect to an SQL server as well as Salesforce. So without hopping from one console to another, I can send queries to both SQL and Salesforce. I can load the data on the data frames and join them to analyze in an integrated environment. Figure 3 shows a very simple code example to fetch data from both sources and merge them.

```
title: Investigation of discrepancy between SQL and Salesforce
author: "Daigo Tanaka"
date: "January 21, 2015"
output: html_document
```{r, echo=FALSE}
library(RPostgreSQL)
library(RForcecom)
# Connect to Salesforce
sf_session = rforcecom.login(sf_username, sf_password, sf_instance_url,
                             sf_api_version)
# Connect to PostgreSQL
drv = dbDriver("PostgreSQL")
con = dbConnect(drv, host=pg_host, dbname=pg_dbname, port=pg_port, user=pg_user,
                password=pg_password)
# Get all active accounts on Salesforce
sf_accounts_query =
    "SELECT * FROM Account WHERE Active__c=true"
sf_accounts = rforcecom.query(sf_session, sf_accounts_query)
# Get all accounts on SQL
sql_accounts = paste(
    "SELECT * FROM app_account WHERE active=true"
sql_accounts = rforcecom.query(sf_session, equipments_w_account_query)
# Merge accounts
accounts = merge(sf_accounts, sql_accounts, by=Id, all=FALSE)
## Active records
Number of active accounts in Salesforce is `r nrow(sf_accounts)`.
Number of active accounts in SQL is `r nrow(sql_accounts)`.
Number of active accounts in both databases `r nrow(accounts)`.
```

Figure 3: R has packages to fetch data from various sources. R-markdown lets us document as we analyze for reproducibility.

**Documentation is the key. The key is to do it painlessly.** In the course of data analysis, we can easily make mistakes in writing complex queries. A wrong query leads to wrong data. Wrong data leads to the wrong conclusion. It is very important that we can revisit the steps of the analysis later and get it peer-reviewed.

It is therefore very important that the queries we write do not get lost and that we have clear documentation for others to review and to reproduce the analysis. However, I find documenting to be very hard if I do it as an independent task after the main task is done. I easily forget the details of the thought process. I lose momentum for executing the task once I think I have seen the result.

R markdown lets us write R code for data analysis and use the result to compose the narrative of the analysis

in the same text file. As long as I leave a note for each steps of data retrievals and transformations, I can produce a report with a push of a button. For example, the R markdown file in Fig. 3 would produce a report that looks like this:

Investigation of discrepancy between SQL and Salesforce

Daigo Tanaka January 21, 2015

**Active records** Number of active accounts in Salesforce is 1251. Number of active accounts in SQL is 1328. Number of active accounts in both databases 1237.

All the data (i.e. the number of accounts) is "soft-coded" and compiled upon the conversion from R markdown to a human readable document format (e.g. HTML, PDF, and etc). The whole text in R markdown is the source code to reproduce the analysis and generate the report. So, R markdown lets us document everything we do as we do it.

Lessons learned In enterprise solutions, we often have to deal with the data from multiple sources such as SQL, Salesforce, logs, NoSQL, and so on. Having to hop from one console to another just to retrieve the data is disruptive to the analytical thought process. As the analysis gets complex, we will make mistakes in writing queries and code to transform data. Leaving the code to reproduce and document the steps of analysis is critical.

- 1. Have an integrated data analytics environment that can load and merge data from multiple sources.
- 2. Do not use graphical use interface (i.e. mouse clicks and spreadsheet program) as it does not leave clear steps of the data retrieval and transformation at the end of the analysis.
- 3. Find a tool that documents the analysis at least semi-automatically for reproducibility and have the analysis peer-reviewed.

#### Conclusions

In this article, I advocated the thought framework of Data Driven Debugging.

When troubleshooting a complex system or doing a data intensive analysis, we should try reducing the hypothesis space with evidence (e.g. logs) before jumping into code when we troubleshoot a complex system.

In enterprise solutions, we should be ready to deal with data from multiple sources. Having an integrated environment reduces the disruption to the analytical thought process. As the analysis gets more complex, it is also crucial to get our analysis peer reviewed. Tools such as R-markdown to semi-automatically document as we run the ad-hoc analysis facilitates the process.

### Acknowledgements

Thanks to dzs for correcting grammar and giving suggestions.