The DataScience Data Engineering Test

A parallelized solution suggestion (in Python)

Three years ago, Jordan Halterman[1] uploaded to the Data Science Inc. GitHub account[2], a small battery of tests. These tests were aimed to evaluate the abilities of a Data Engineering candidate.

I´ve found that several companies are using them as a technical basis for hiring processes. And, when I read the first of them, I decided to try it for myself.

It is apparently simple, but devil is always in details, and the problem became an interesting example of ETL.

The problem: GitHub can´t parse the file

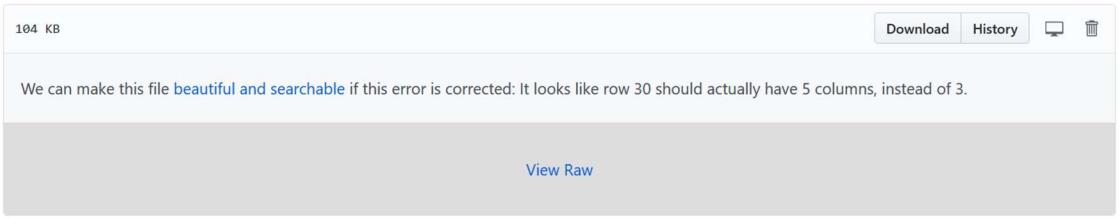
Jordan said that the test came *from a real-world experience working on client data*. The question? **An apparently simple .tsv file that couldn´t be read by normal tsv readers and parsers**:

Here´s the file:

```
id      first_name      last_name       account_number  email
1       Addison Marks   196296  ornare.lectus@et.edu
2       Dakota  Garza   409025  scelerisque@Praesentluctus.edu
3       Basia   Wolfe   637720  Aliquam@nullaIntegerurna.com
4       Germaine        Campbell        826846  id.magna@viverraMaecenas.ca
5       Lenore  Pennington      345284  aliquam@Integer.edu
```

Innocent, eh? Five normal fields separated by tabs. Nothing special.

But, as the test description says: *If you clicked on the link above you may notice that GitHub complains about errors in the file*:



Row 30 should actually have 5 columns, instead of 3. Why?

[1] Distributed Systems Engineer at @OpenNetworkingFoundation, *kuujo* in GitHub
[2] *https://github.com/datascienceinc*

Provided that this dataset is in SQL-style (a NoSQL piece of information wouldn´t have had any problem like that); and the rows must have the same quantity of fields, this is really an issue.

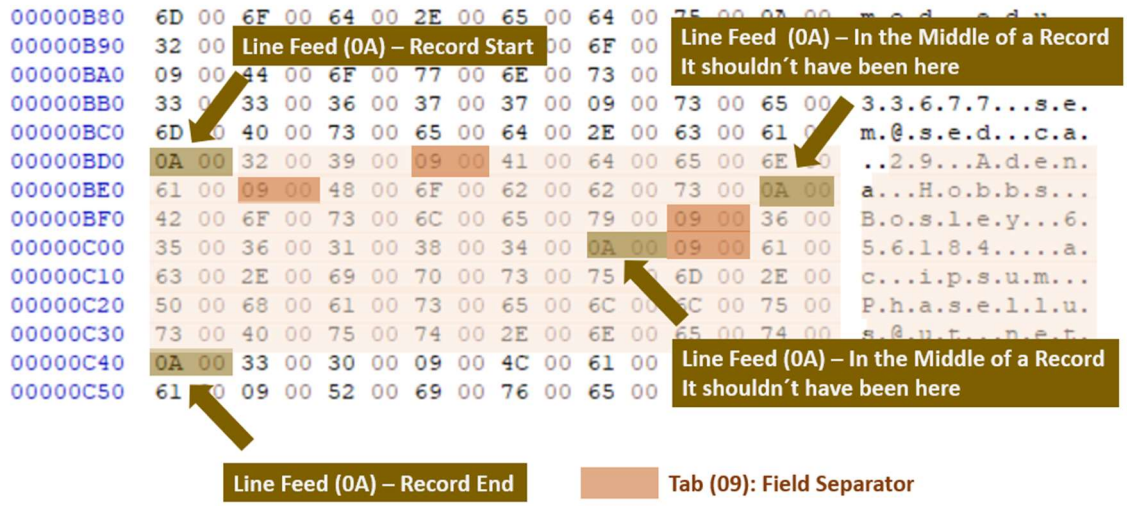
As the author states: *as GitHub cannot accurately parse the file, neither can our Hadoop, Pig, or data warehouse utilities*.

The real problem: line feeds in the middle of some fields

Well, let´s look into the file, that strange row 30.

```
28      Ivory   Downs   133677  sem@sed.ca
29      Adena   Hobbs
Bosley  656184
        ac.ipsum.Phasellus@ut.net
30      Laura   Rivera  270464  nascetur.ridiculus.mus@Donecnibhenim.org
31      Clinton Vincent 677802  adipiscing.lacus.Ut@atlibero.edu
```

Oops! The record number 29 (Adena Hobbs Bosley) is splitted in three lines. Why? Let´s see it with a hexadecimal editor:



Every record starts with a Line Feed character (0Ah) and finishes with the next Line Feed. Every field in the record starts with a Tab character (09h) and finishes with the next Tab.

As you can see, two Line Feeds gate-crashed in the middle of the record. Virtually every parser in the world understands Line Feed as a record end and would break the 29th record in three small pieces

There are other similar Line Feed characters out of place, and their records are broken in a similar way.

The challenge is now clear

So, it´s imperative to do something with this file. The proposed challenge is *write a simple script to transform data.tsv into a properly formatted tab-separated values (TSV) file that can be read by any standard CSV/TSV parser*.

A clean solution: you get a damaged file and you give a correct file back. And the process can continue for that file.

And the main formatting suggestion for the solution is:

> **Fields that contain tab characters should be given back quoted.**

It is consistent with the spirit of losing as less data as possible from the original file. Even those damned line feeds must be kept. If the misplaced line feeds are enclosed in quotes, its destructive capacity is overturned.

And those strange 00s?

Until now, we have been focused in the central problem and set aside collateral questions. Let´s deal with them.

```
00000B80  6D 00 6F 00 64 00 2E 00 65 00 64 00 75 00 0A 00  m.o.d...e.d.u...
00000B90  32 00 38 00 09 00 49 00 76 00 6F 00 72 00 79 00  2.8...I.v.o.r.y.
00000BA0  09 00 44 00 6F 00 77 00 6E 00 73 00 09 00 31 00  ..D.o.w.n.s...1.
00000BB0  33 00 33 00 36 00 37 00 37 00 09 00 73 00 65 00  3.3.6.7.7...s.e.
00000BC0  6D 00 40 00 73 00 65 00 64 00 2E 00 63 00 61 00  m.@.s.e.d...c.a.
00000BD0  0A 00 32 00 39 00 09 00 41 00 64 00 65 00 6E 00  ..2.9...A.d.e.n.
00000BE0  61 00 09 00 48 00 6F 00 62 00 62 00 73 00 0A 00  a...H.o.b.b.s...
00000BF0  42 00 6F 00 73 00 6C 00 65 00 79 00 09 00 36 00  B.o.s.l.e.y...6.
00000C00  35 00 36 00 31 00 38 00 34 00 0A 00 09 00 61 00  5.6.1.8.4.....a.
00000C10  63 00 2E 00 69 00 70 00 73 00 75 00 6D 00 2E 00  c...i.p.s.u.m...
00000C20  50 00 68 00 61 00 73 00 65 00 6C 00 6C 00 75 00  P.h.a.s.e.l.l.u.
00000C30  73 00 40 00 75 00 74 00 2E 00 6E 00 65 00 74 00  s.@.u.t...n.e.t.
00000C40  0A 00 33 00 30 00 09 00 4C 00 61 00 75 00 72 00  ..3.0...L.a.u.r.
00000C50  61 00 09 00 52 00 69 00 76 00 65 00 72 00 61 00  a...R.i.v.e.r.a.
```

As it can be seen, every byte in the file is escorted by a 00h character. Let´s make way to another small difficulty: the original file is encoded in UTF-16LE, and the corrected file must be delivered in UTF-8.

After checking the even positions in the file, I concluded that all of them were 00h. So, if I would remove them in the result file, no information would be lost.

A simple solution

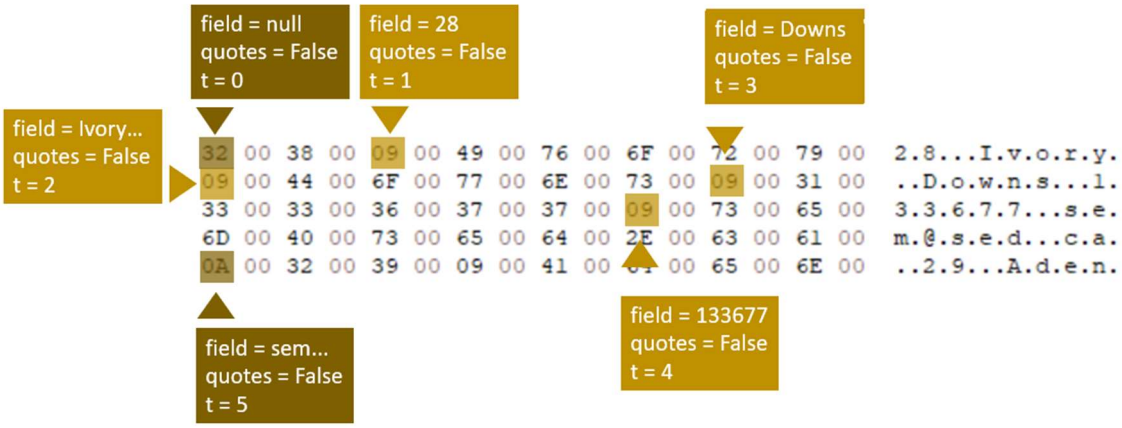
A simple solution would be going down the file, byte to byte, and, for each 0Ah found, checking if it is in the middle of a record. In that case, we should enclose all the field in quotes.

How do I do that? Using a “quotes” field. If the 0Ah is found in the middle of a field, this parameter is raised to True. And, when the record end is found, the whole record is wrapped into quotes.

A normal record

Let´s see a normal record. At starting, the *field* value is null, and the *t* value is 0. *t* will act as the field number (five fields or each record: *id*, *first_name*, *last_name*, *account_number*, *email*).

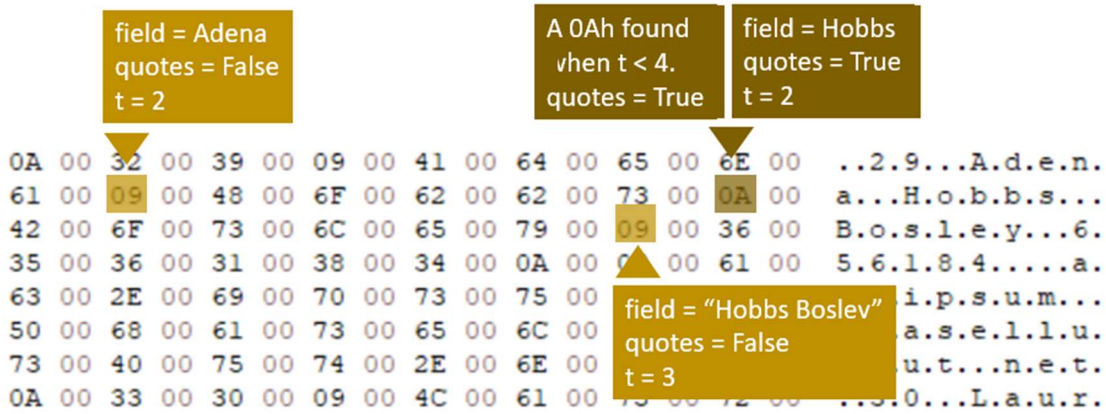
Each time a 09h (\t, tab) is found, the field values are changed.



When a 0Ah (line feed) is found, the record is finished. And the record values are reset.

A defective record

Here is a record with problems. The 0Ah is found when *t* = 2. So, the line feed is in the middle of the record, not at its end. The “quotes” parameter is raised and the whole field is wrapped into quotes.



And, voilà, mission accomplished. The offending 0Ah is now locked up among quotes.

```
6D 40 73 65 64 2E 63 61 0A 32 39 09 41 64 65 6E  m@sed.ca.29.Aden
61 09 22 48 6F 62 62 73 0A 42 6F 73 6C 65 79 22  a."Hobbs.Bosley"
09 22 36 35 36 31 38 34 0A 22 09 61 63 2E 69 70  ."656184.".ac.ip
73 75 6D 2E 50 68 61 73 65 6C 6C 75 73 40 75 74  sum.Phasellus@ut
2E 6E 65 74 0A 33 30 09 4C 61 75 72 61 09 52 69  .net.30.Laura.Ri
```

The file can now be easily read by a normal tables parses. R for example:

26	26	Nathaniel	Ruiz	327166	magna.Sed.eu@arcuMorbisit.com
27	27	Kimberley	Parks	319377	vestibulum.lorem.sit@eueuismod.edu
28	28	Ivory	Downs	133677	sem@sed.ca
29	29	Adena	Hobbs Bosley	656184	ac.ipsum.Phasellus@ut.net
30	30	Laura	Rivera	270464	nascetur.ridiculus.mus@Donecnibhenim.org
31	31	Clinton	Vincent	677802	adipiscing.lacus.Ut@atlibero.edu
32	32	aretha	Torres	278324	scelerisque.neque.sed@consequatdolor.ca

Parallelization

The code for this simple script is in gitHub. But it hasn´t really a great relevance. There are thousands of more complicated problems solved each day by the legion of programmers giving support to our digitalized world.

The real challenge comes with the parallelization

> *For bonus points, ambitious candidates can parallelize their algorithm.*

I´m not actually a candidate, but I´m ambitious. Especially regarding Parallelization, Big Data, Machine Learning and all this exciting stuff.

And for helping people who starts with parallelization is because I have written this article. Not for experts and specialists, but for learners needing a concrete case. An interesting case beyond the parallelization of any easy “hello world” file.

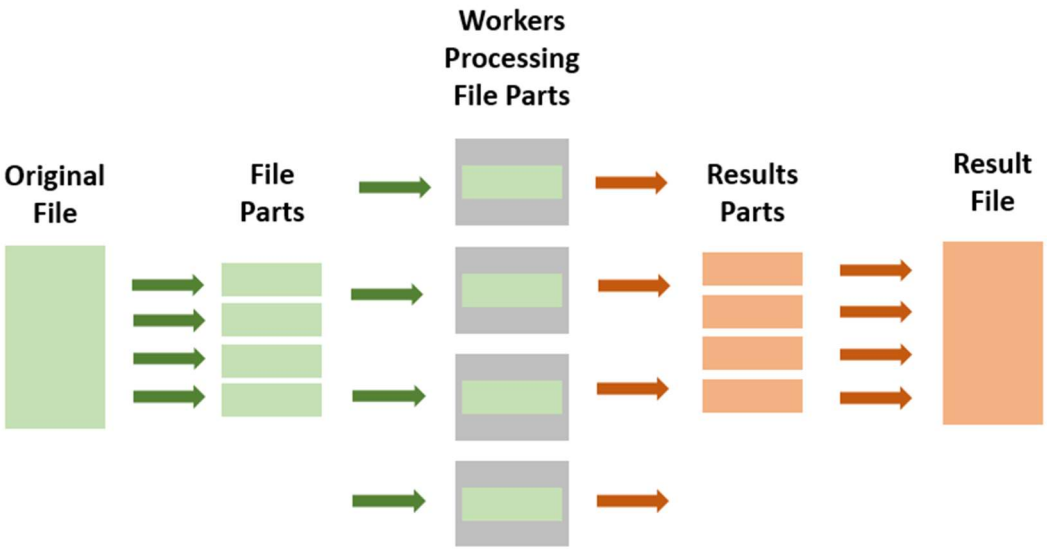
The behaviour of parallelization

There are a lot of approximations to this technique, but I have had in mind the successful and ubiquitous framework of Apache Spark.

The idea for this process is to take a dataset (the given file, in this case), divide them in several parts, and give each of these parts to a worker, node, container, … in a separate CPU.

The same script is executed in each of those containers. The results for each container are gathered and merging all these parts gives a final result.

Advantages? Time. The time a CPU walks for all over the file is divided by the number of workers doing the same job …. *In parallel*. 😊



The division problem

We can then suppose that this process will work fine, at an incredible speed, delivering exact results without losing data.

Everything would be as the text says:

> *Given an arbitrary byte position and length, the algorithm cleans a portion of the full data set and produces a unique TSV output file.*
>
> *Concatenating the outputs of multiple processes should result in a well-formed TSV file containing no duplicates.*

But here we have the key of the problem:

> *It's important to note that the arbitrary position may not necessarily be the start of a new line.*

Each complete record in each worker will be treated and "fixed" using the already provided script. But, what about the records cut by the division?

Each part of the original file will start with the tail of a record and will end with the head of another. The script won't be able of count the fields for decide if one of them contains a spurious 0Ah.

The broken fields

Let´s suppose that the record #201 is split among two parts of the file. One of them would end that way:

```
n.a.s...c.a...2.
0.1...K.u.a.m.e.
..C.o.l.e...7.7.
4.
```

id: **201**
first_name: **Kuame**
last_name: **Cole**
account_number: **774**

And other of then would start like that:

```
2.8.8...p.e.n.a.
t.i.b.u.s...e.t.
@.d.o.l.o.r...o.
r.g...2.0.2...K.
```

account_number: **288**
email: **penatibus.et@dolor.org**

The worker that treats each fragment doesn´t know anything about the other part of the record. Then, it will be necessary to set a process that reunite the record parts after processing and fixing them. If we don´t treat these fragments in an appropriate manner, we could lose information, or have partial records.

The solution

Before proposing my own solution to the problem, I would like to comment some small considerations:

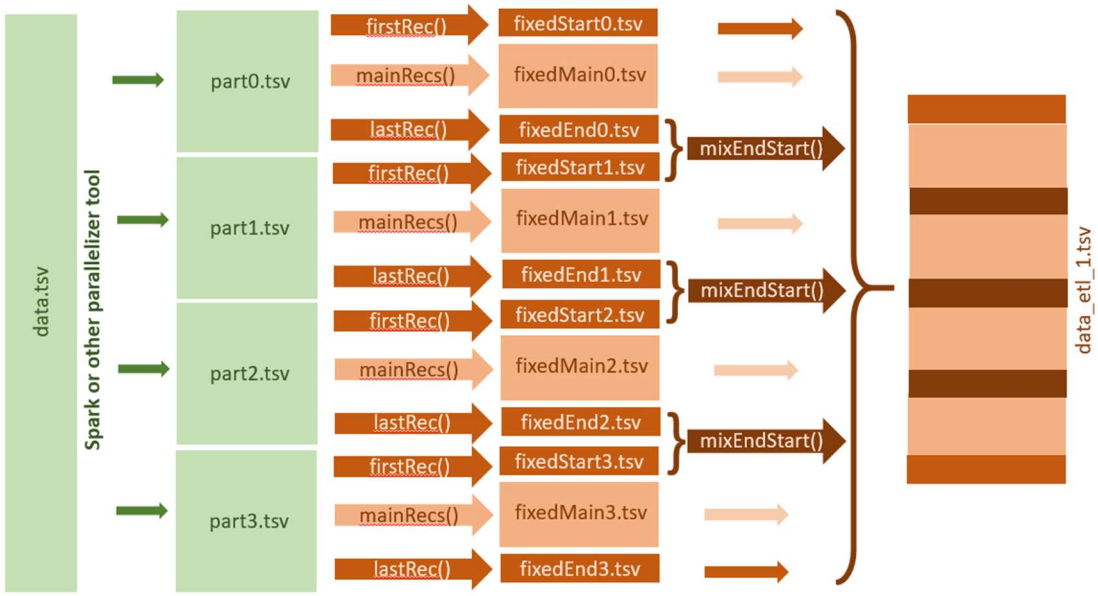
- I could have divided the process in a set of Spark transformations and processes. But I have tried to make a unique script. Several transformations could provide more time and computational cost that a unique process split into a set of parallel workers.
- I have explored other approaches like regular expressions, or separator for Spark. But, none of them seemed to be simple. And I want to use this article for to help people who are starting with parallelization and ETL.
- It is supposed that every record has an *id* field, that field is consecutive, and there is no gap in the series. (This is important for the function *firstRec()* to infer the value of the *id* field for that record.
- There is a script in Python that is easily adaptable to any parallelization framework or technique.

For this sample, I have divided the original data set (*data.tsv*) in five equal parts (*part0.tsv*, *part1.tsv*, *part2.tsv*, *part3.tsv*, *part4.tsv*). Each one of this parts is the feed for each worker in a

parallelized system. In this case, and for testing purposes, the same script treats each part sequentially.

Intermediate files

For each part (0 to 4), the original part (*partX.tsv*) is read by three functions: *firstRec()*, *mainRecs()*, *lastRec()*, that create three intermediate files: *fixedStartX.tsv*, *fixedMainX.tsv*, *fixedEndX.tsv*.



The half records, united

Every resulting file is fixed. The problem now is the half-records in the starting and the ending of each part.

For each of them, the *lastRec()* and *firstRec()* functions have created files (*fixedEndX.tsv*, *fixedStartX.tsv*), that have the same *id*. For the *fixedEndX*.tsv file, when a field is unknown, it is included, but with a *Null* value (nothing between the 09h separators). For both, If the field is partial, its part is passed to the file.

Then, for each pair End-Start, we will have two files. For example:

```
32 30 31 09 4B 75 61 6D 65 09 43 6F 6C 65 09 37  201.Kuame.Cole.7
37 34 09 0A                                      74..
```

Is a *fixedEndX.tsv* file. And:

```
32 30 31 09 09 09 32 38 38 09 70 65 6E 61 74 69  201...288.penati
62 75 73 2E 65 74 40 64 6F 6C 6F 72 2E 6F 72 67  bus.et@dolor.org
0A                                               .
```

Is a *fixedStartX.tsv* one.

As we can see, the first of them is formed by the fields *id=201*, *first_name=Kuame*, *last_name=Cole*, and a partial *account_number* field, *774*.

The *End* file is formed by the fields *id=201*, *first_name=Null*, *last_name=Null*, a partial *account_number* field, with value *288*, and *email=penatibus.et@dolor.org*.

The function *mixEndStart()* takes each field for both records and concatenate them. The *Null* fields don´t add anything, and the partial field is restored. The *id* field has a special treatment for not to be duplicated.

How to know the id field when you don´t have the beginning of the record?

There is a small piece of code that takes several fields ahead (15 in the proposal), checks if there are two consecutives numbers separated by five fields, and guess that the id number is the minor of them minus one:

<code>

The final reassembling

Once the *fixedXXX.tsv* files ready, it´s necessary to reassemble them sequentially. Each parallelization framework has its own procedures or, as in the example shown, the own script will do this task at the end. The result: the original file with those offending internal 0Ah enclosed in quotes, without loss of data.

Only a simple approach to show the parallelization problem

Of course, there will be a lot of more effective, clever, cleaner or sophisticated approaches to this problem. My aim was only to give to the beginners of parallelization an interesting problem and suggest how to solve it. In such a way that they can touch the main problem of the code independence for each of the nodes or workers.

The code for this test is available in *https://github.com/antonio-eslava/data-engineering-test*