Trabajo práctico n. 2

Middleware y coordinación de procesos: Yelp reviews analysis

7574 - Sistemas Distribuidos I

Docentes

- Pablo D. Roca
- Ezequiel Torres Feyuk
- Guido Albarello

Alumno

- Rozanec, Matías 97404 mrozanec@fi.uba.ar

https://github.com/rozanecm/7574-tp2

2C 2020
Facultad de Ingeniería - Universidad de Buenos Aires

Índice

Trabajo práctico n. 2 **0**

Middleware y coordinación de procesos: Yelp reviews analysis **0**

7574 - Sistemas Distribuidos I **0**

Índice **1**

Introducción **2**

Nomenclatura 2

Ejecución **2**

Demo **2**

Resultados obtenidos **2**

Diagramas **3**

DAG 3

Vista física 4

Vista lógica 6

Vista de procesos 7

Vista de desarrollo 9

Dificultades encotradas **9**

Sincronismo de EOT 9

Conclusión **10**

Anexo **11**

Resultados 11

Resultados obtenidos 11

Resultados reales 12

Introducción

En el presente informe se presenta la arquitectura diseñada para resolver el problema planteado, consistente en procesar un stream de datos de forma distribuida. Para la comunicación de los distintos servicios se usaron colas de RabbitMQ.

Ejecución

Toda la ejecución del sistema es controlada mediante los comandos `./run.sh` y `./stop.sh`, extendiendo la funcionalidad de los ejemplos de RabbitMQ provistos por la cátedra.

Demo

Para la ejecución del TP durante la demo se ha optado por realizar una grabación de una corrida que se puede ver en el siguiente enlace: https://youtu.be/ndw3RYfqib8

La razón para proceder de esta forma es que normalmente la corrida entera del sistema ha llevado alrededor de 10 minutos. Con ningún otro programa abierto se ha llegado a 8 minutos.

Si bien para la demo se podría realizar una ejecución en vivo procesando solamente una parte del dataset, el hecho de que haya una videollamada en curso probablemente afecte los tiempos de ejecución, y se corre el riesgo de que igualmente se extienda más allá del tiempo esperado. Además está el agravante de que no se puede comparar resultados, porque obviamente darán distinto a los esperados.

De todas maneras, si el docente corrector así lo exigiera, se puede realizar una corrida reducida.

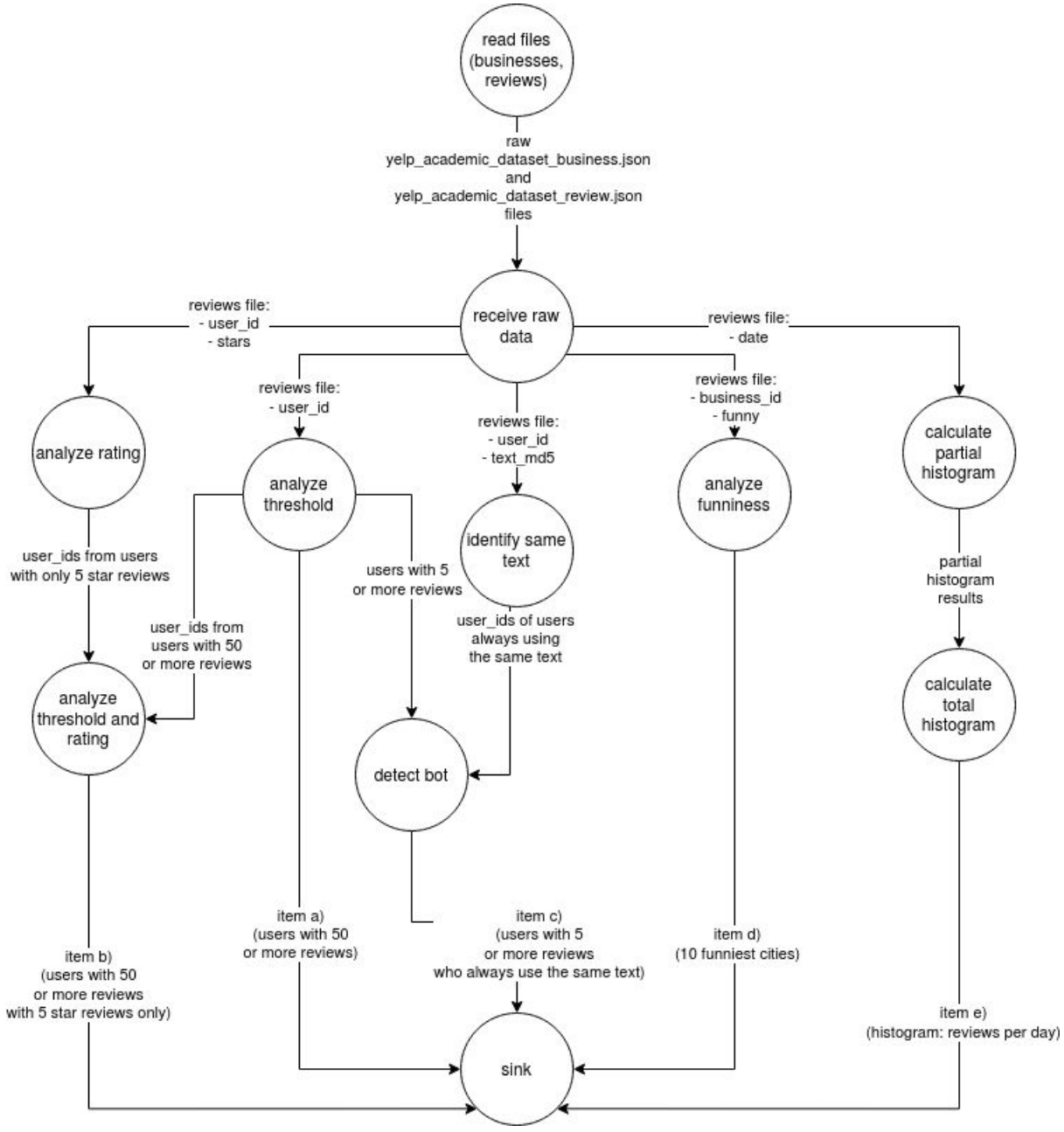
Resultados obtenidos

Los resultados obtenidos se muestran completos, tal como los recibe el cliente, en el anexo. Los mismos han sido contrastados contra los resultados del Notebook que compartió el prof. Pablo Roca, y que un compañero logró correr. Dichos resultados son también incluidos en el anexo.

Diagramas

A continuación se presentan los diagramas que permiten entender el sistema desarrollado.

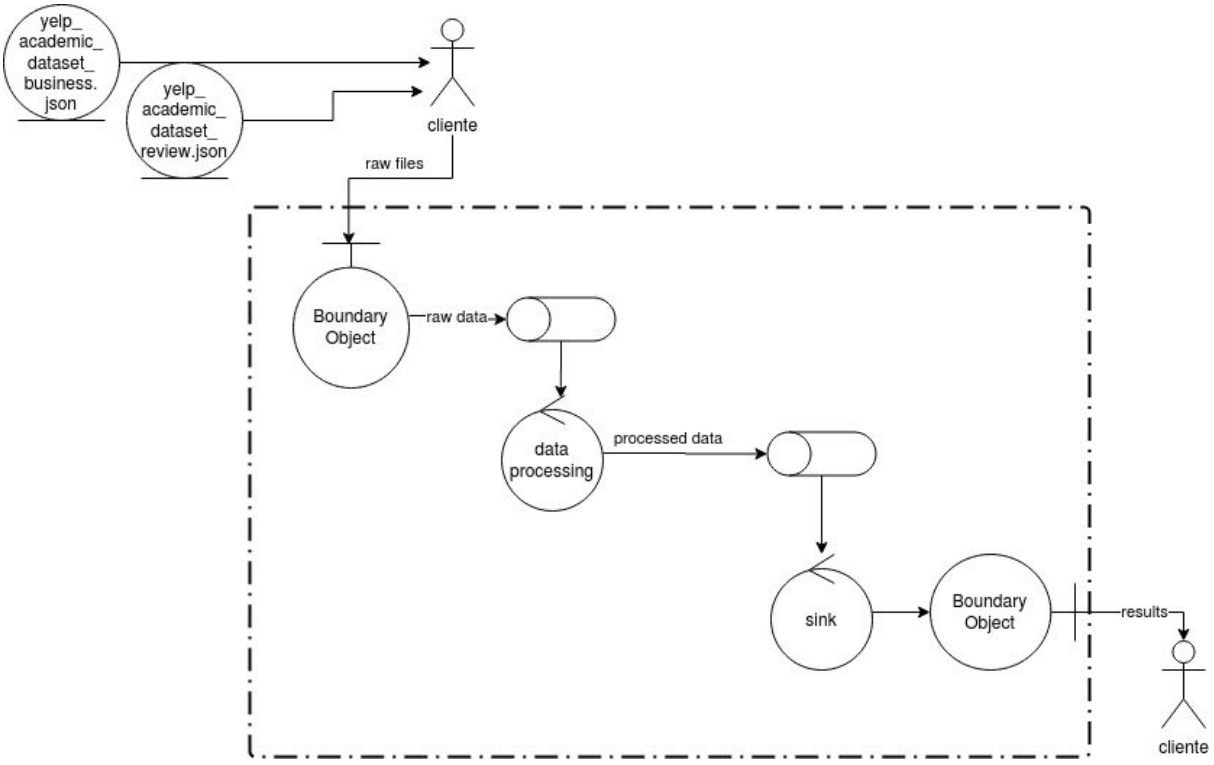
DAG



En este diagrama se pueden ver las distintas acciones que hay que ir ejecutando para llegar a computar los 5 resultados solicitados. Se puede ver cómo las tareas más complejas que exigen considerar dos condiciones distintas, pueden ser fácilmente distribuidas en nodos independientes, y cómo el nodo Threshold analyzer puede ser utilizado para tres de los resultados finales. Esto es así porque ese nodo guarda la frecuencia de cada usuario, y finalmente transmite los usuarios que han superado el threshold establecido para cada una de las consignas.

Observar también que ya desde el principio el sistema se deshace del texto, transmitiendo solamente un hash del mismo, que es suficiente para la tarea solicitada.

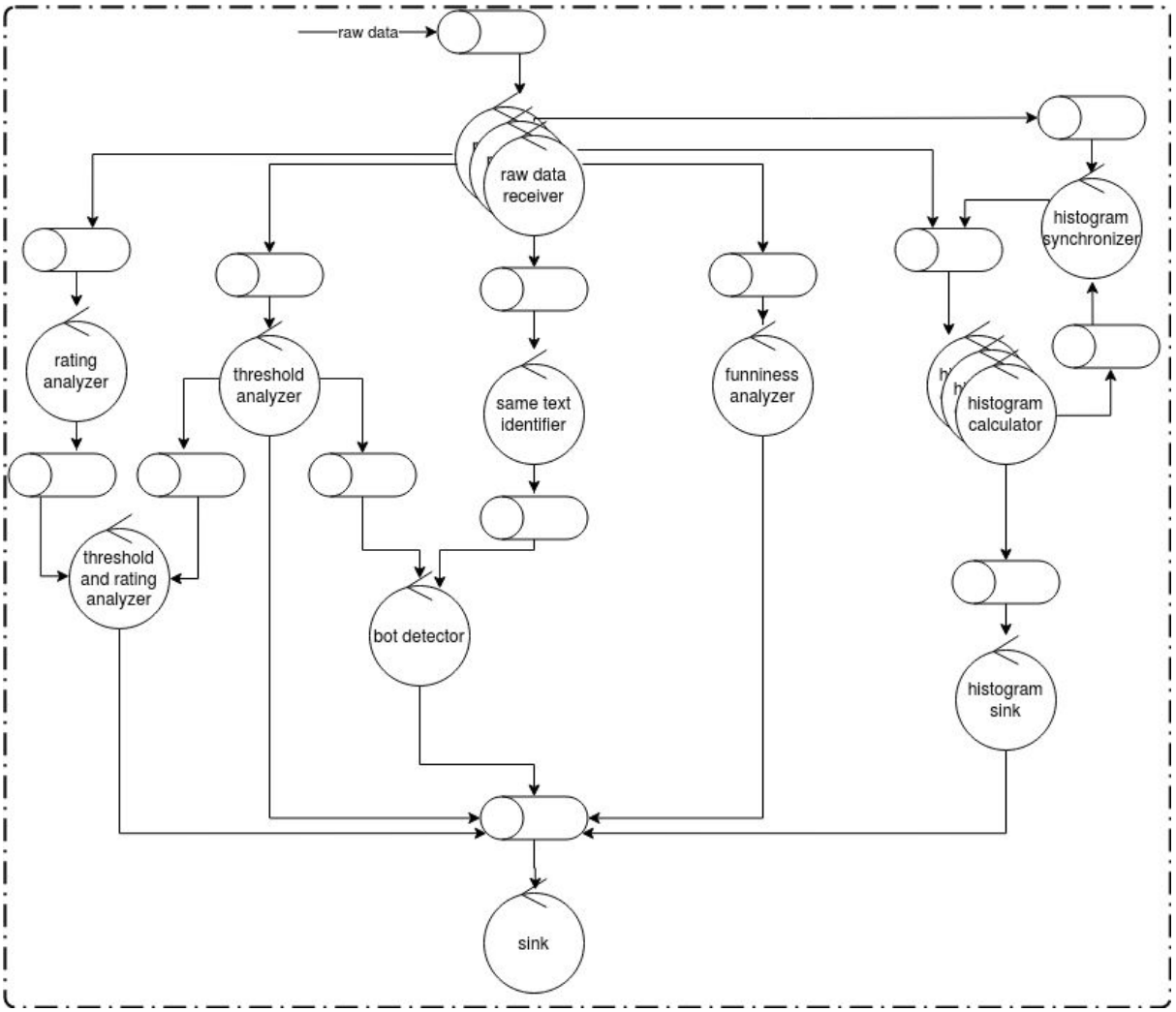
Vista física



En el diagrama adjunto se puede ver que toda la información entra al sistema por medio del cliente, que es quien lee los archivos con información de businesses y reviews. Una vez que la información entra al sistema de forma cruda, i.e. sin procesar, el nodo que la recibe, el Raw data receiver o simplemente Receiver, truncará esa información quedándose solamente con la información que es necesaria para cada nodo. Esto es de vital importancia, ya que de esta forma se evita que por el sistema circule más información de la necesaria.

Notar que hay un solo canal de comunicación entre el cliente y el sistema. Esto es posible porque el cliente entrega información en un diccionario, indicando con una key si la información enviada es de los businesses o de los reviews. Como no solo no es necesario, sino que tampoco debe suceder que se envíe información de los reviews antes que de los businesses, esta secuencialidad en esta parte del TP no afecta en nada a la performance del mismo.

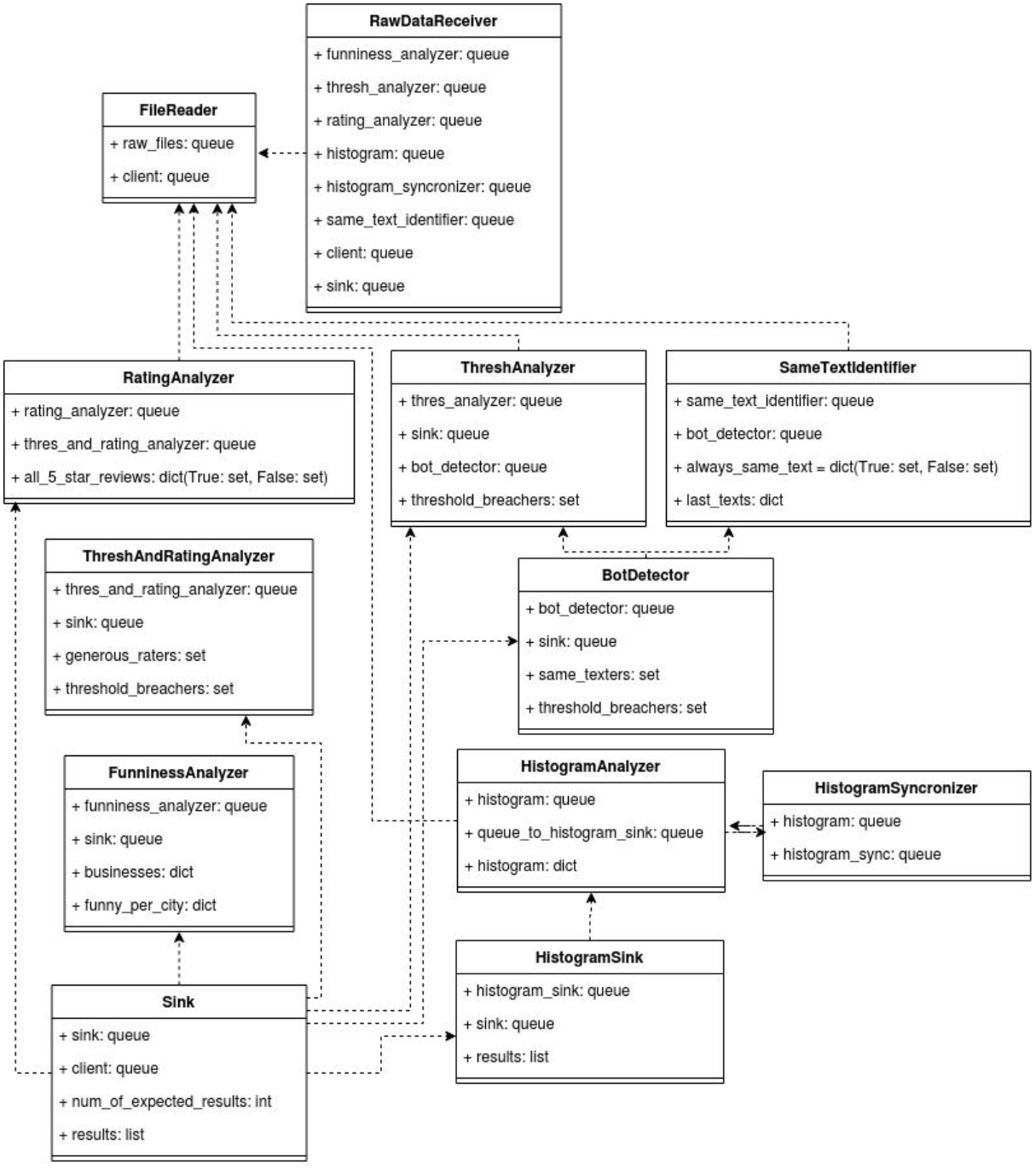
Aquí se encuentra el primer punto de distribución del TP: cada bulk de información puede ser recibido y procesado independientemente del resto, por lo tanto ese nodo en particular es infinitamente escalable.



El otro nodo que es infinitamente escalable es el del cálculo de histogramas: se ha encontrado durante el desarrollo del TP que el procesamiento de la fecha para obtener un objeto datetime que pueda ser interpretado para armar el histograma era suficientemente exigente como para el nodo se vuelva un cuello de botella. Como el procesamiento no requiere de contexto de otros mensajes, este nodo puede ser escalado también. Finalmente, cuando se hayan terminado de recibir todos los datos, cada uno de los nodos con histogramas parciales enviará estos resultados al Histogram sink, que se encarga de juntar los histogramas, transmitiendo así al sink un único histograma con toda la información requerida.

Finalmente, si bien el cliente que está arriba de todo y abajo de todo en el diagrama es el mismo, se lo ha dibujado dos veces por claridad.

Vista lógica



Como se puede observar, no hubo composiciones entre las distintas clases. Cada entidad definida juega más bien un rol bastante independiente, donde la relación entre las distintas partes se da exclusivamente mediante mensajes. Sin embargo, para no dejar las clases sin ningún tipo de conexión, se ha optado por representar las distintas dependencias que hay entre las entidades, por más que las mismas no representen composiciones estrictamente hablando. Se logra así obtener un pantallazo de la complejidad de cada clase y las interrelaciones que debe manejar cada una de ellas.

En este diagrama se pueden apreciar las colas que maneja cada parte. Contrastando este diagrama con el anterior, de Robustez, se empieza a formar una idea más clara del sistema armado.

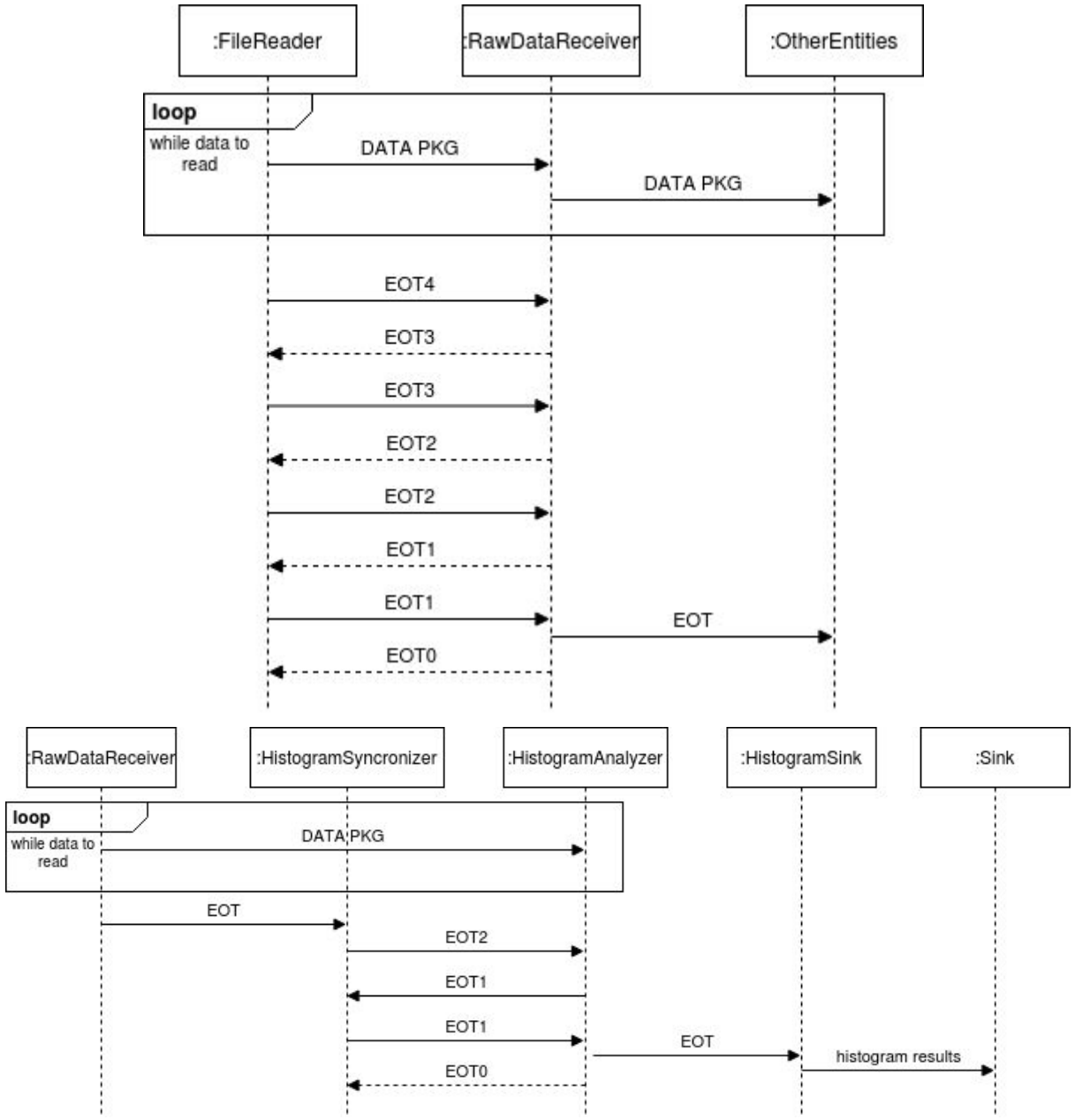
Vista de procesos

En este primer diagrama de secuencia se muestra cómo se maneja la multiplicidad de Receivers. Esto es muy importante, ya que todos los Receivers deben estar al tanto del fin de transmisión de información de los archivos. Además estos Receivers deben transmitirle a los otros nodos que se ha alcanzado el fin de la transmisión, para que puedan informar cada uno sus resultados al sink, y finalmente éste también al cliente.

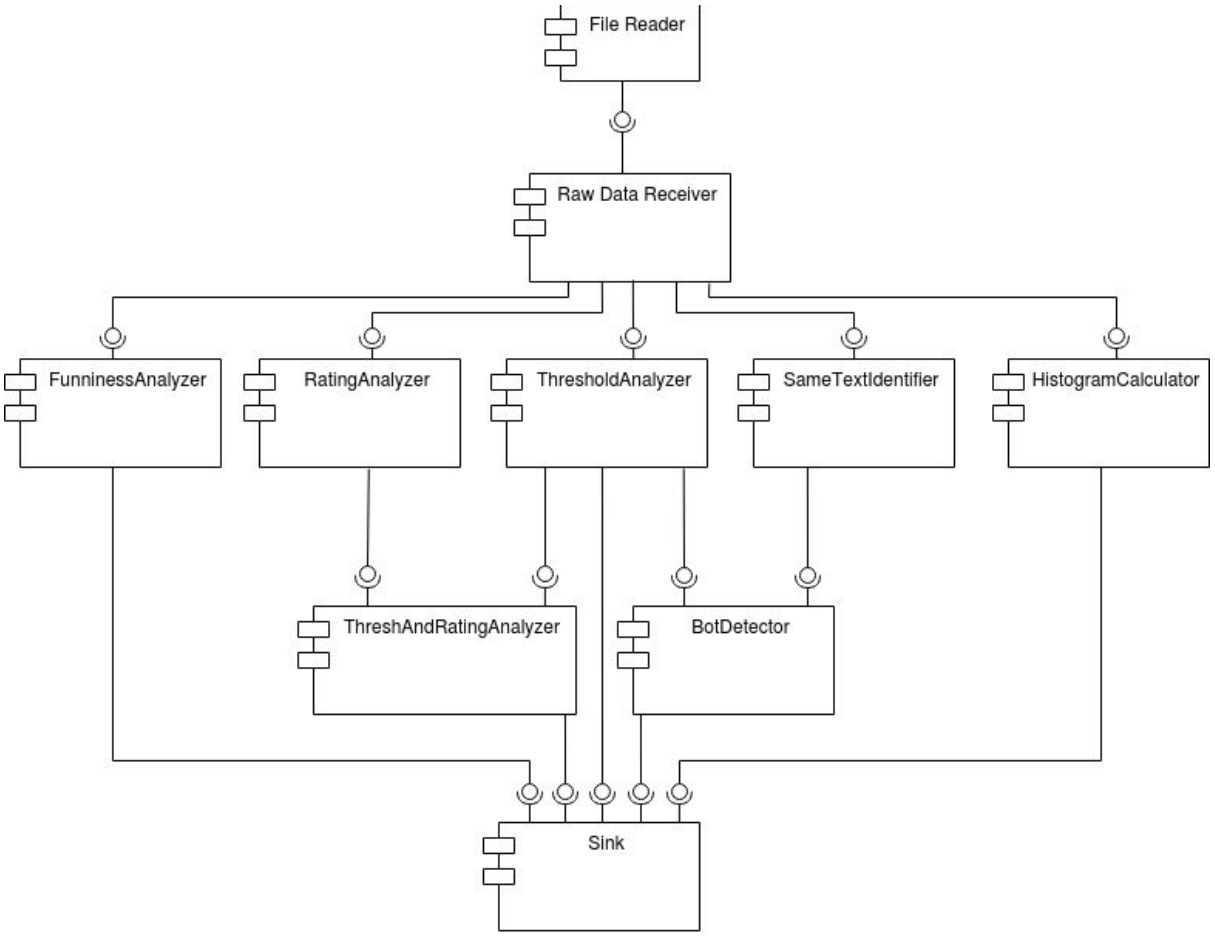
La forma en que se logra esta sincronización es mediante el conocimiento, de parte del cliente, de la cantidad de lectores que hay. De esta manera, se envía un mensaje de fin de transmisión EOT[n] (End of Transmission), donde n indica la multiplicidad de Receivers. Cada Receiver le responderá al cliente con otro mensaje EOT, pero con n-1 anexado al final. Este mensaje será enviado nuevamente a los Receivers, hasta que se reciba el mensaje EOT0, indicación de que ya todos los receivers están al tanto del fin de transmisión.

El último Receiver en recibir el mensaje de fin de transmisión es el encargado de transmitir el mensaje al resto de las entidades. Si bien en la mayoría de los casos esto sucede de forma bastante directa, en el caso de los Histogramas resulta un poco más complejo por el manejo de la multiplicidad (hay varios nodos a los que habría que avisarles).

Para lidiar con esto, se le envía el mensaje de fin de transmisión al nodo Histogram Syncronizer, un nodo que se encarga de realizar la misma dinámica que se describió entre cliente y Receivers, y que se puede observar en detalle en el segundo diagrama de Secuencia.


:FileReader
:RawDataReceiver
:OtherEntities
loop
while data to read
DATA PKG
DATA PKG
EOT4
EOT3
EOT3
EOT2
EOT2
EOT1
EOT1
EOT
EOT0
:RawDataReceiver
:HistogramSyncronizer
:HistogramAnalyzer
:HistogramSink
:Sink
loop
while data to read
DATA PKG
EOT
EOT2
EOT1
EOT1
EOT
histogram results
EOT0

Vista de desarrollo



En este diagrama se puede observar a grandes rasgos cuáles son los componentes que componen el sistema desarrollado, y cómo son las dependencias/comunicaciones entre ellos.

Dificultades encotradas

Sincronismo de EOT

Programar correctamente esta parte presentó un desafío interesante y distinto del resto del TP. En principio, el uso de colas durante todo el TP resultó muy natural, sin mayores complicaciones: pensando en qué se quería lograr y armando el DAG de antemano, sumado a las discusiones en clases, el desarrollo de todo eso fue bastante lineal. El tema del sincronismo en cambio exigió más atención al detalle, exigió crear canales en ambos sentidos entre algunos nodos, cosa que si bien parece trivial, se vuelve torpe rápidamente. Fue también, indudablemente, una de las partes que donde más se aprendió.

Conclusión

El presente trabajo práctico me obligó a enfrentarme a nuevos paradigmas y formas de pensar y resolver los problemas. Siendo la primera materia en que trabajo con una arquitectura “plenamente distribuida” que deba ser escalable, muchos conocimientos previos tuvieron que adaptarse de alguna forma a esto.

Si bien finalmente logré implementar una solución que cumple con los requerimientos solicitados, siento que hay mucho lugar para mejorar. ¡Espero que esas potenciales mejoras se vean plasmadas en los siguientes trabajos prácticos!

Anexo

Resultados

Resultados obtenidos


```
[
  {
       "funniest cities": [
       "City: Las Vegas, funny reviews count: 513206",
       "City: Phoenix, funny reviews count: 184937",
       "City: Toronto, funny reviews count: 112887",
       "City: Scottsdale, funny reviews count: 92798",
       "City: Charlotte, funny reviews count: 64070",
       "City: Tempe, funny reviews count: 49241",
       "City: Henderson, funny reviews count: 49042",
       "City: Pittsburgh, funny reviews count: 46446",
       "City: Cleveland, funny reviews count: 40974",
       "City: Mesa, funny reviews count: 37819"
       ]
  },
  {
       "Users with 50+ reviews": 15955
  },
  {
       "Days of the week histogram": {
       "Monday": 1183732,
       "Tuesday": 1112199,
       "Wednesday": 1124486,
       "Thursday": 1080564,
       "Friday": 1090099,
       "Saturday": 1184179,
       "Sunday": 1245863
       }
  },
  {
       "chronic generous raters": [
       "xBxmaLiSLXN68Gqj_zdjkQ",
       "nkhu7NjlIEimaJ-QD8S3SA",
       "aYV4_aVwexS-Zn34k_P0VQ",
       "8JwSmvviX2dEAgaPRZ70nQ",
       "MTl8QNjnenumWaORnXhing",
       "uQ2w3yMEYxcspPrZLKObVw",
       "2bC-dBYN48wrmN-j0Bv-Jw",
       "trGQ2nySSedAbXaaHhcmUQ",
       "caUyy1kvh-MxntoKoJr6kg",
       "aGwONlR86ERk450Gp6Ih5A",
       "2-PD6df20ge-k9SPHmcxiw"
       ]
  },
  {
       "likely to be bots": [
       "TV5s5qQKgMGoECfDLGdTmQ",
```

```
      "k8Hw_ua1KjCPGVkhGOk7ew",
      "9II6XRbZAf2koLK1lCjz6A",
      "ImyvYbCzWhoYnbJzEvGGgw"
      ]
  }
]
```

Resultados reales

```
top_10_funny_cities
```

	city	qty
396	Las Vegas	513206
687	Phoenix	184937
934	Toronto	112887
848	Scottsdale	92798
138	Charlotte	64070
925	Tempe	49241
322	Henderson	49042
703	Pittsburgh	46446
155	Cleveland	40974
496	Mesa	37819

```
len(reviews_qty_gt_50)
```

```
15955
```

```
day_of_week
0    1183732
1    1112199
2    1124486
3    1080564
4    1090099
5    1184179
6    1245863
dtype: int64
```

	total_qty	starts_5_qty
2-PD6df20ge-k9SPHmcxiw	54	54.0
2bC-dBYN48wrmN-j0Bv-Jw	59	59.0
8JwSmvviX2dEAgaPRZ70nQ	195	195.0
MTI8QNjnenumWaORnXhing	55	55.0
aGwONlR86ERk450Gp6lh5A	55	55.0
aYV4_aVwexS-Zn34k_P0VQ	60	60.0
caUyy1kvh-MxntoKoJr6kg	56	56.0
nkhu7NjllEimaJ-QD8S3SA	64	64.0
trGQ2nySSedAbXaaHhcmUQ	66	66.0
uQ2w3yMEYxcspPrZLKObVw	80	80.0
xBxmaLlSLXN68Gqj_zdjkQ	54	54.0

```
users_same_texts_always
```

[23]:

user_id	total_qty	text_hash	same_texts_qty
9ll6XRbZAf2koLK1lCjz6A	11	971fa13e286683f4b300cc725b870d1b7c64121b	11
lmyvYbCzWhoYnbJzEvGGgw	6	1fdcf5f12420fef03a93c9bf1b3fdb0423e1c847	6
TV5s5qQKgMGoECfDLGdTmQ	5	1f808a2dac075af57c54b92f462fe587dfbaa9d3	5
k8Hw_ua1KjCPGVkhGOk7ew	6	b832916b96512c121321a55b4da456d3cf1ed532	6