Universitat de les Illes Balears

Escola Politècnica Superior

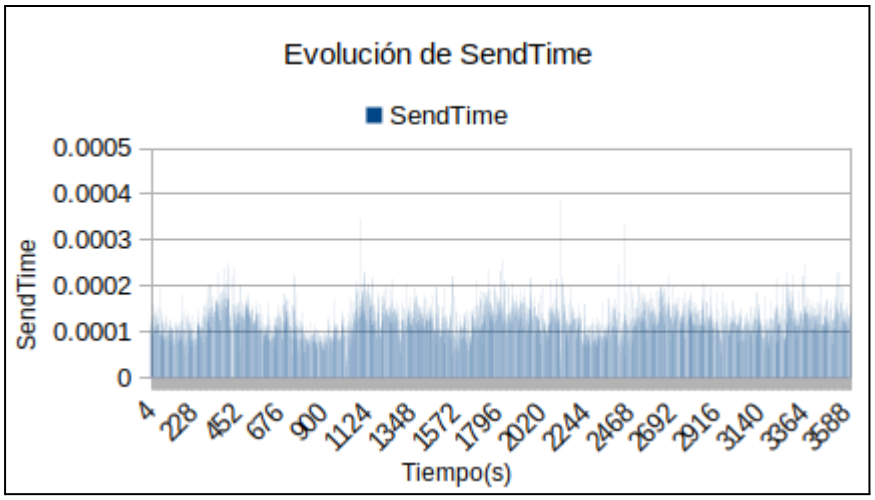
21719 - Avaluació del Comportament de Sistemes Informàtics.

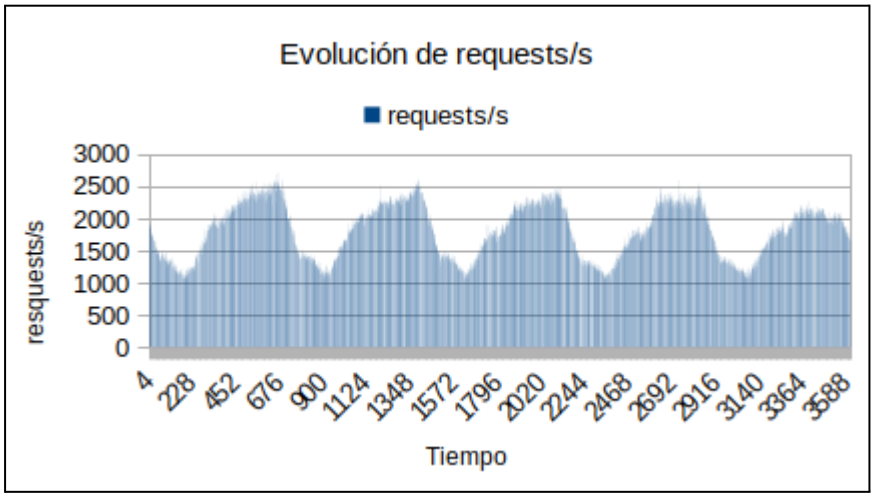
Práctica 3 : Tema 6 - Caracterización de la Carga

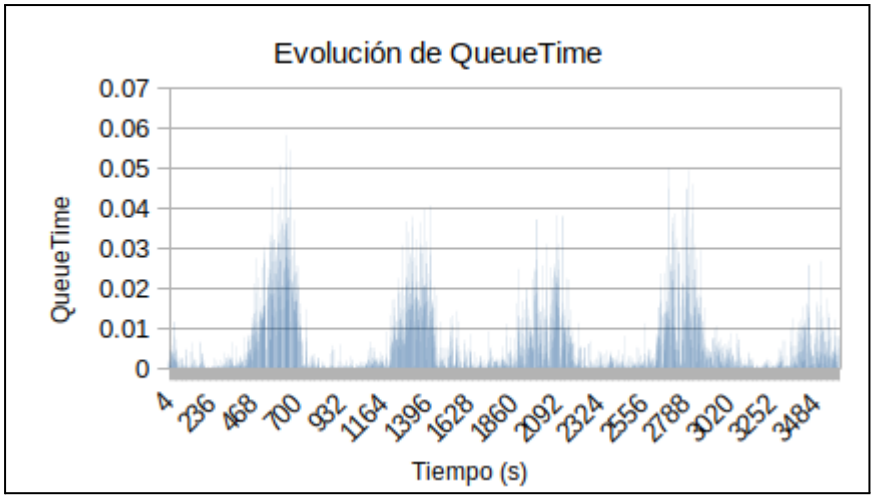
Khaoula Ikkene
Grupo 102
khaoula.ikkene1@estudiant.uib.cat

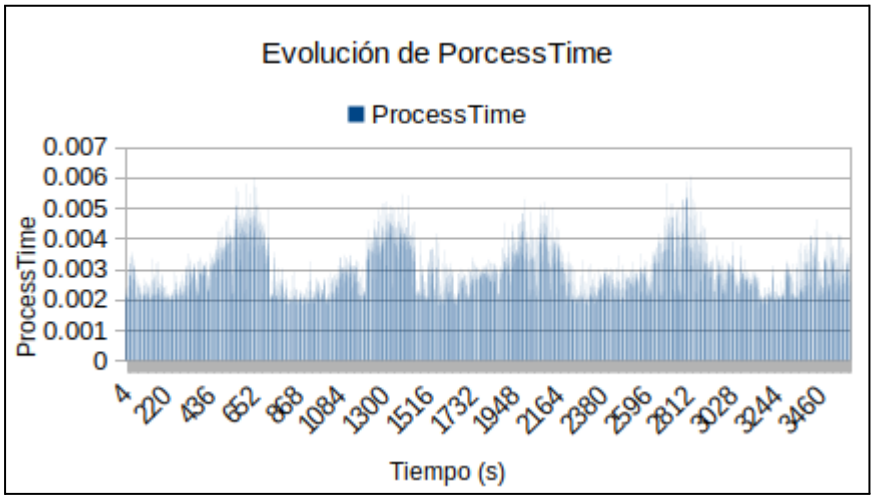
1. ¿Hay alguna característica especial en la carga proporcionada? Explícala con detalle

Al representar los datos de nuestras cuatro columnas (requests/s, ProcessTime, QueueTime, SendTime) con el tiempo podemos observar claramente que siguen patrones cíclicos. Las siguientes gráficas muestras dicho comportamiento.









Los patrones observados en los datos son cíclicos y no estacionales, dado que el periodo de repetición de cada patrón se mide en segundos (las muestras se recogen cada dos segundos).

En detalle:
El comportamiento del sendtime tiene un patrón que se repite aproximadamente cada 23 minutos.
El queuetime muestra un patrón repetitivo aproximadamente cada 12 minutos.
El processtime presenta un patrón recurrente cada 38 minutos.
Se observa también que los incrementos en todas las gráficas coinciden aproximadamente. Es decir, cuando aumenta el queuetime, también aumentan el processtime y el sendtime.

Es relevante destacar que al generar una representación gráfica tridimensional de las tres columnas (processtime, queuetime y sendtime), se evidencia que los valores de sendtime son considerablemente más bajos en comparación con los de processtime y queuetime. Además, en la gráfica, solo se visualizan los valores de sendtime. Esto sugiere que las solicitudes son enviadas al cliente en un intervalo de tiempo generalmente más corto que el tiempo que pasan en la cola de espera o siendo procesadas.

El comportamiento del queuetime es notablemente volátil, ya que alcanza valores máximos muy rápidamente y luego disminuye a valores mínimos en poco tiempo.

El processtime tiene un comportamiento estale. Cuando aumenta el tiempo de espera en la cola (queuetime), también aumenta el tiempo de procesamiento de las peticiones (processtime). Esto es razonable, ya que un aumento en las peticiones entrantes incrementa el tiempo de espera en la cola, lo que conduce a una mayor competencia por los recursos y a la sobrecarga del sistema, y esto puede disminuir el tiempo de procesamiento de las peticiones.

2. Aplicando el algoritmo con 100 iteraciones y agrupando los datos en 3 clases, ¿qué resultados se obtienen? Muéstralo gráficamente.

Aplicamos el algoritmo de k-means con 100 iteraciones, agrupando los datos en 3 clusters y utilizando la distancia euclidiana. Los resultados obtenidos son los siguientes:

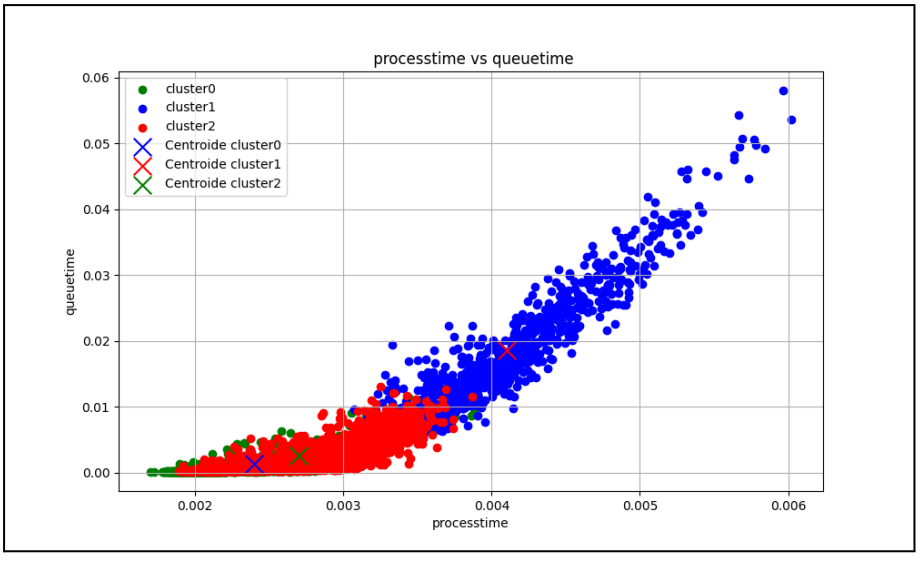
```
kMeans
======

Number of iterations: 16
Within cluster sum of squared errors: 121.67591302208115

Initial starting points (random):

Cluster 0: 1164.542701,0.002005,0.000324,0.000091
Cluster 1: 1580.850698,0.002421,0.000211,0.000133
Cluster 2: 1207.601196,0.002403,0.000141,0.00013

Missing values globally replaced with mean/mode

Final cluster centroids:
                        Cluster#
Attribute     Full Data         0          1          2
               (3600.0)   (1368.0)    (827.0)   (1405.0)
=================================================================
requests/s     1804.311 1334.0545 2286.4463  1978.393
processtime      0.0029    0.0024    0.0041    0.0027
queuetime        0.0057    0.0013    0.0185    0.0026
sendtime         0.0001    0.0001    0.0001    0.0001
```

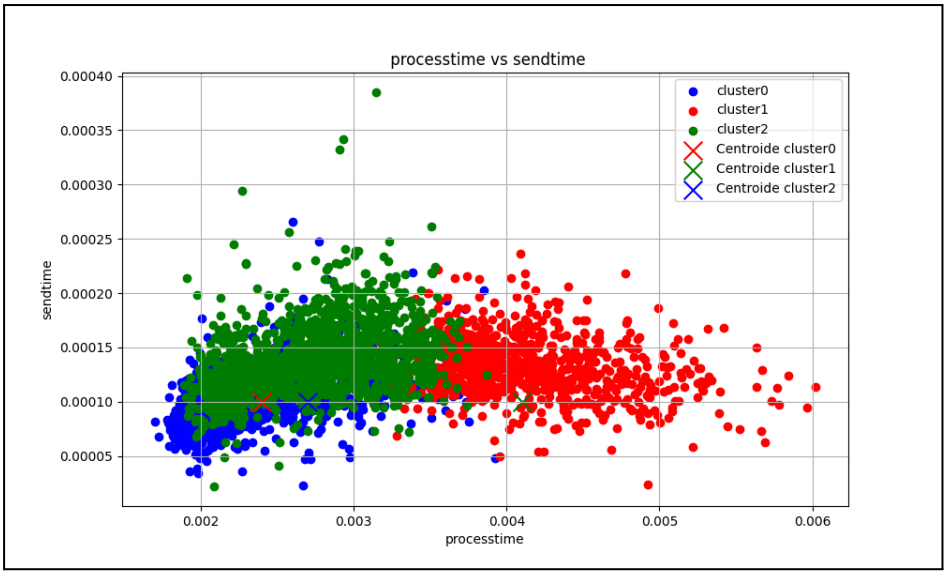
```
Time taken to build model (full training data) : 0.05 seconds

=== Model and evaluation on training set ===

Clustered Instances

0       1368 ( 38%)
1        827 ( 23%)
2       1405 ( 39%)
```

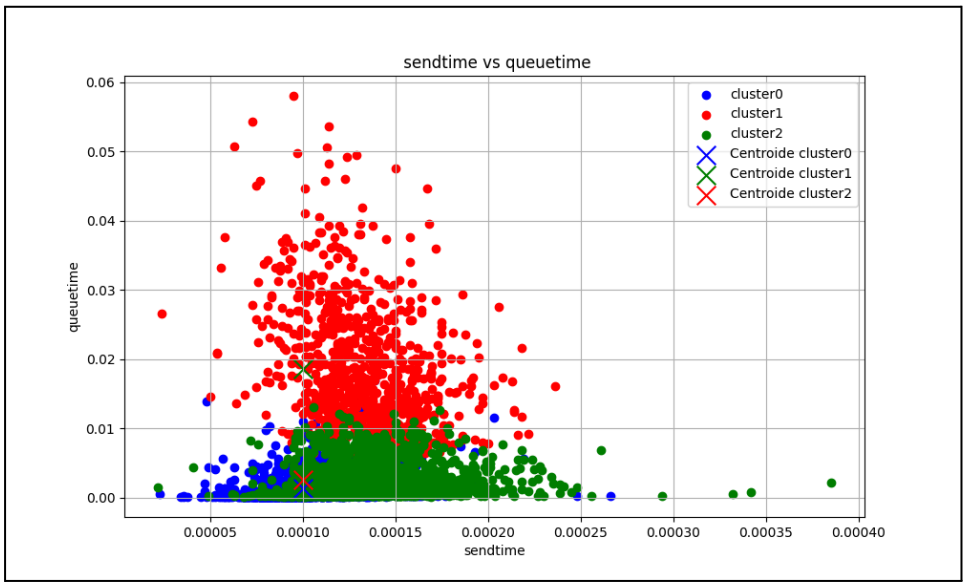
Podemos verificar la corrección de estos grupos sumando los porcentajes de los clusters creados: 38% + 23% + 39% = 100%. Por lo tanto, este clustering es el más adecuado para la serie de datos que tenemos.

La herramienta WEKA proporciona varias visualizaciones; sin embargo, las gráficas de requests/s no ofrecen información relevante sobre el comportamiento de la solicitud en el momento monitorizado. Por lo tanto, únicamente presentaremos las gráficas correspondientes a las otras tres columnas: processTime, queueTime y sendTime. En particular, mostraremos las gráficas de las combinaciones processTime-queueTime, processTime-sendTime y queueTime-sendTime, ya que las otras serían redundantes al invertir los ejes de abscisas y ordenadas o al representar la misma información en ambos ejes.

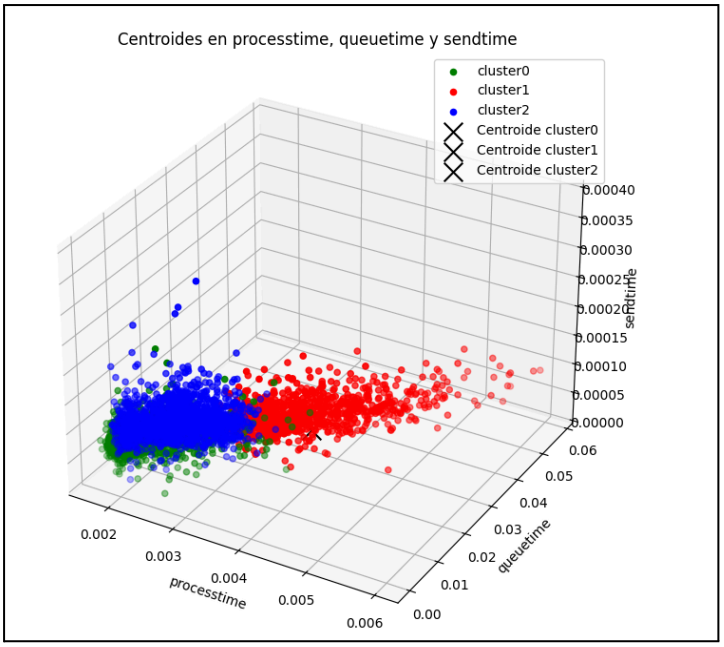
A continuación, presentaremos las gráficas con los centroides dibujados, con la ayuda de un pequeño programa en python.





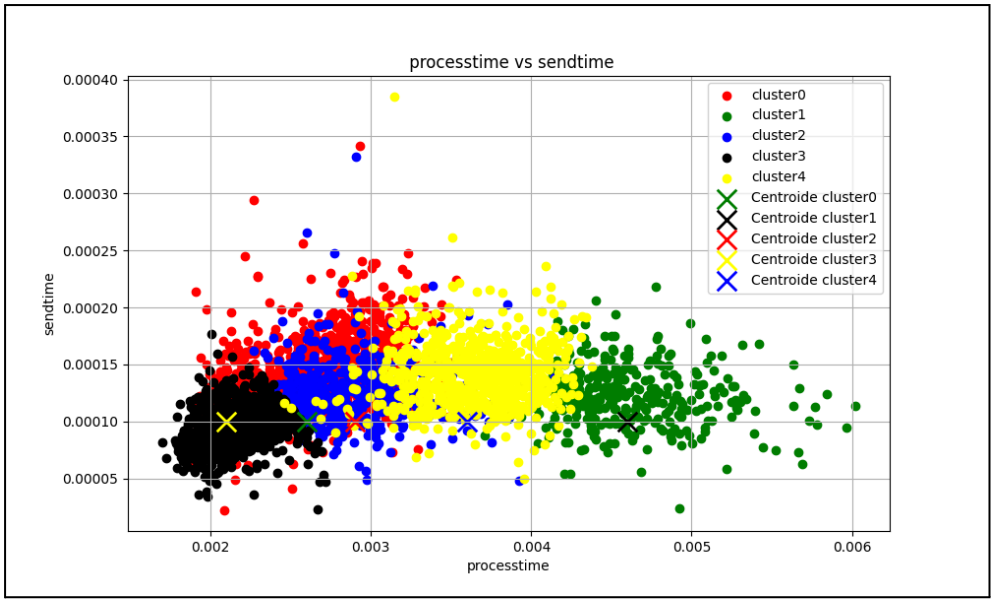


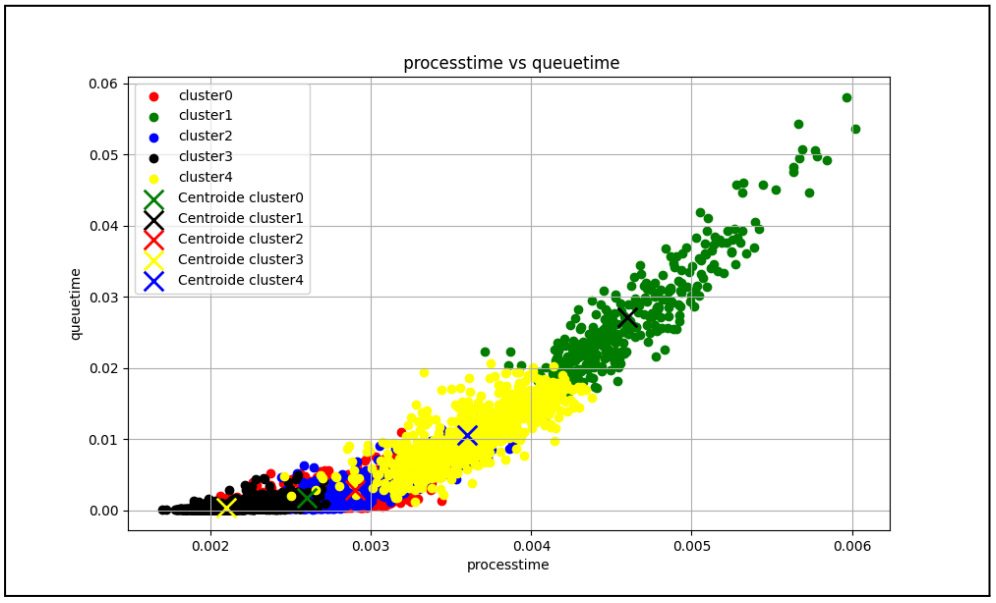
En el espacio tridimensional, obtenemos la siguiente representación gráfica. Aunque los centroides no son muy visibles en esta visualización, es evidente que los clusters se solapan significativamente. Esto se debe a que los valores que toman están, en su mayoría, dentro del mismo rango.

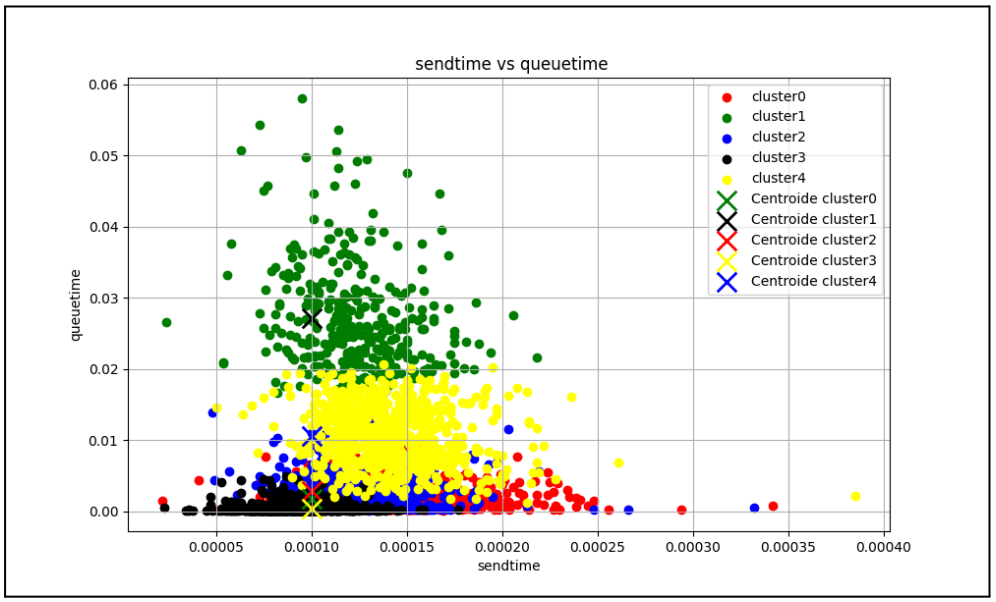


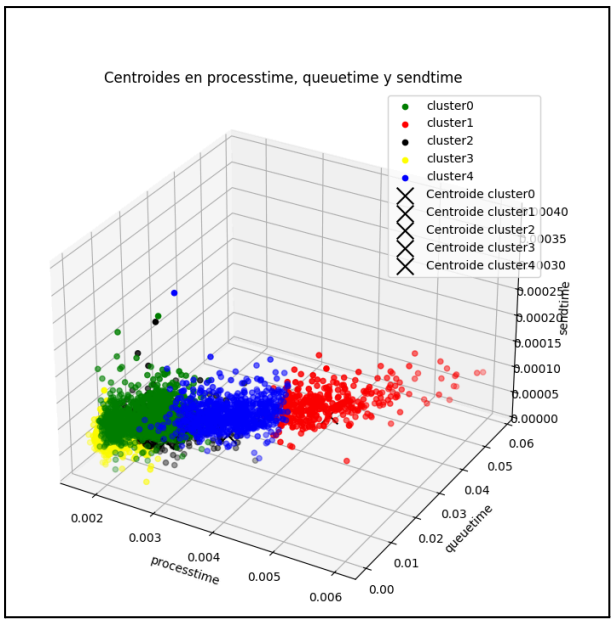
3. Con el mismo número de iteraciones y agrupando los datos en 5 clases, ¿qué resultados se obtienen? ¿Cómo difieren de los anteriormente obtenidos?

```
Missing values globally replaced with mean/mode

Final cluster centroids:
                        Cluster#
Attribute     Full Data         0         1         2         3         4
               (3600.0)  (1163.0)   (335.0)   (487.0)   (876.0)   (739.0)
=============================================================================
requests/s     1804.311 1937.5898 2308.9958 1324.0702 1336.0191 2237.3672
processtime       0.0029    0.0026    0.0046    0.0029    0.0021    0.0036
queuetime         0.0057    0.0017    0.0271    0.0029    0.0004    0.0106
sendtime          0.0001    0.0001    0.0001    0.0001    0.0001    0.0001
```


processtime vs sendtime
cluster0
cluster1
cluster2
cluster3
cluster4
Centroide cluster0
Centroide cluster1
Centroide cluster2
Centroide cluster3
Centroide cluster4
sendtime
processtime
0.00040
0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.002
0.003
0.004
0.005
0.006


processtime vs queuetime
cluster0
cluster1
cluster2
cluster3
cluster4
Centroide cluster0
Centroide cluster1
Centroide cluster2
Centroide cluster3
Centroide cluster4
queuetime
processtime
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0.002
0.003
0.004
0.005
0.006


sendtime vs queuetime
cluster0
cluster1
cluster2
cluster3
cluster4
Centroide cluster0
Centroide cluster1
Centroide cluster2
Centroide cluster3
Centroide cluster4
queuetime
sendtime
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0.00005
0.00010
0.00015
0.00020
0.00025
0.00030
0.00035
0.00040

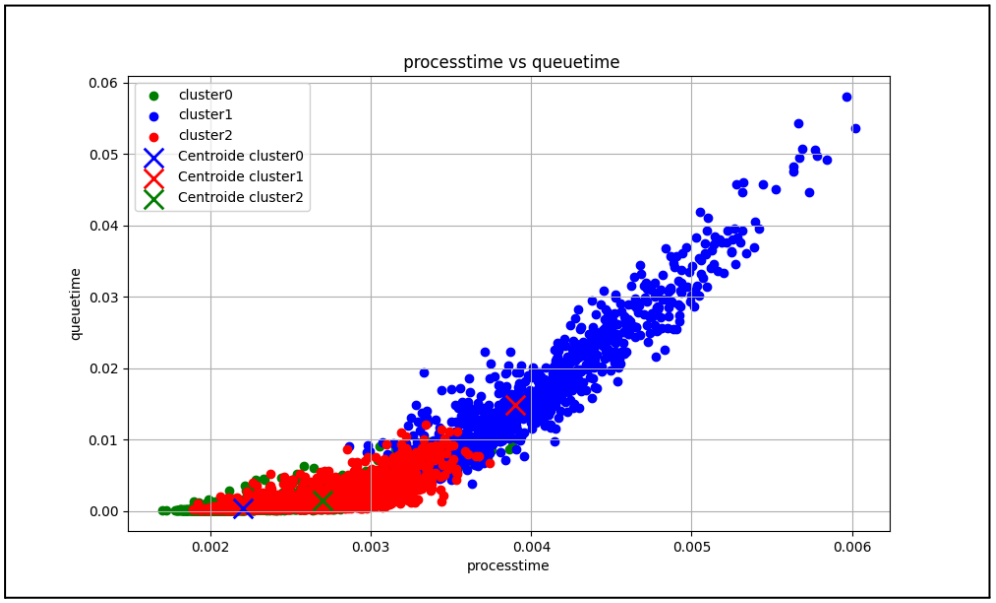


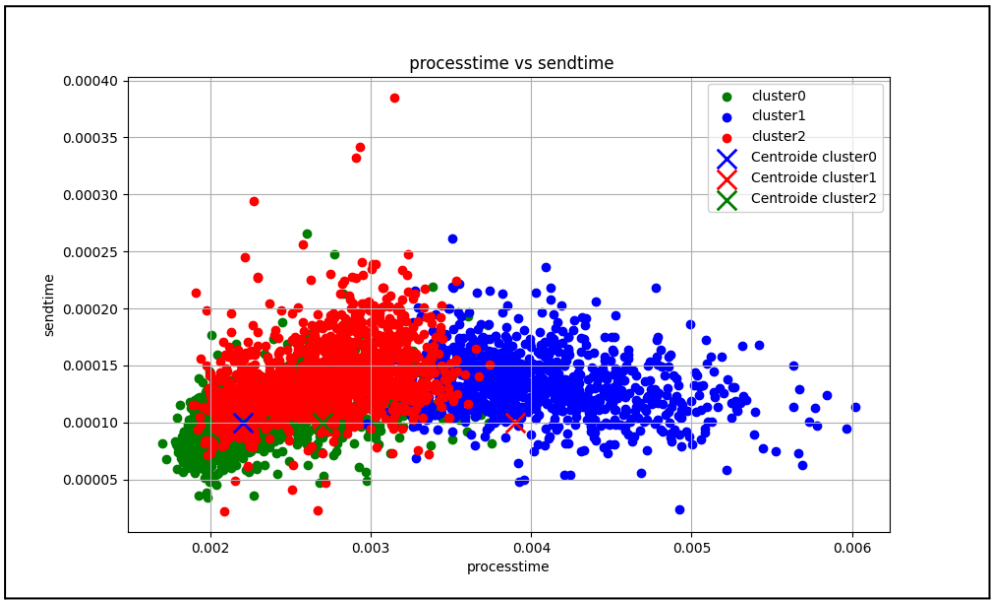
Observamos que con 5 clusters, la clasificación es más detallada y precisa. En las gráficas bidimensionales, los centroides muestran también un comportamiento coherente. Por ejemplo, en la gráfica de sendtime-queuetime, los centroides se alinean con una abscisa x = 0.0001. De manera similar, en la gráfica de processtime-sendtime, los centroides comparten una ordenada y = 0.0001, y esto se debe principalmente a que los centroides de la columna sendtime se mantiene constantes, aunque cambiamos el número de clases de agrupamiento.

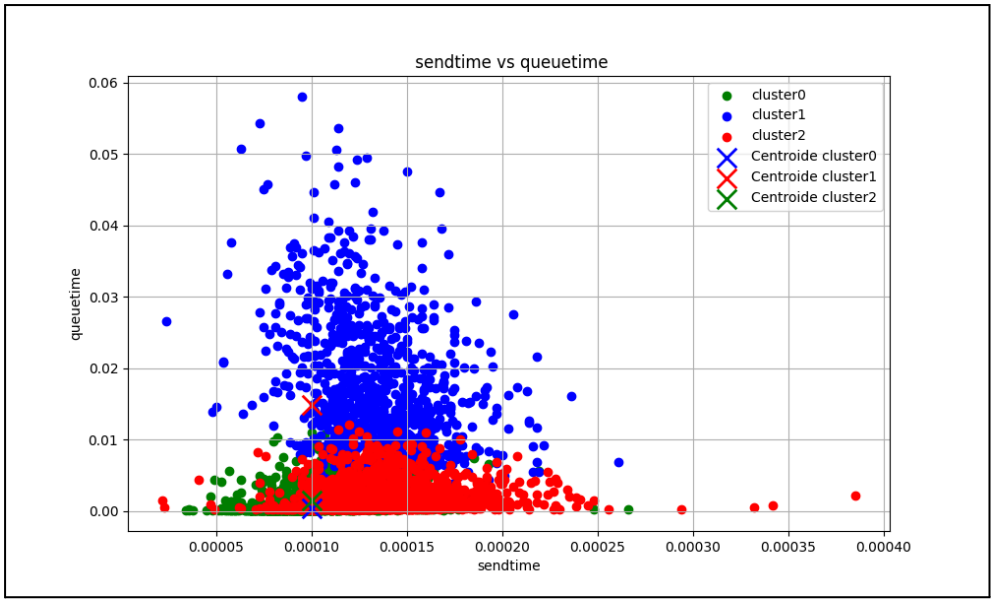
4. Cambiando la distancia de la Euleriana a la Manhattan, vuelve a aplicar el punto 2. ¿Qué diferencias hay al cambiar la distancia? Muéstralo gráficamente

```
Final cluster centroids:
                        Cluster#
Attribute     Full Data         0          1          2
               (3600.0)   (1345.0)    (947.0)   (1308.0)
=========================================================
requests/s    1844.8437 1318.2887 2274.1332 1948.3173
processtime      0.0028    0.0022    0.0039    0.0027
queuetime        0.0018    0.0003    0.0148    0.0015
sendtime         0.0001    0.0001    0.0001    0.0001



Time taken to build model (full training data) : 0.02 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      1345 ( 37%)
1       947 ( 26%)
2      1308 ( 36%)
```

La primera diferencia que notamos es que la suma de las instancias no es 100% (37%+26%+36%=99%).


processtime vs queuetime
cluster0
cluster1
cluster2
Centroide cluster0
Centroide cluster1
Centroide cluster2
queuetime
processtime
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0.002
0.003
0.004
0.005
0.006


processtime vs sendtime
cluster0
cluster1
cluster2
Centroide cluster0
Centroide cluster1
Centroide cluster2
sendtime
processtime
0.00040
0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.002
0.003
0.004
0.005
0.006



Al cambiar a la distancia de Manhattan, notamos un cambio significativo en la posición de los centroides, especialmente en la columna queuetime. Mientras que los centroides de la columna sendtime permanecieron constantes, los de processtime cambiaron apenas en unas pocas unidades, alrededor de 0.0001.

A simple vista, los centroides resultantes al cambiar a la distancia de Manhattan parecen estar más centrados en la densidad del cluster en comparación con los centroides obtenidos anteriormente. Esto podría ser atribuido a la robustez de la distancia de Manhattan frente a valores atípicos, como los que se presentan en nuestros datos.

Es notable que los centroides de la columna sendtime permanecieron invariables, lo cual podría ser resultado de la distribución casi uniforme de los datos en esta columna, como se mencionó previamente, con pocos valores atípicos.

En resumen, como se ha explicado previamente, la presencia de valores atípicos provocará un mayor cambio en la posición de los centroides al usar una distancia u otra.

Anexo

Código en Python para dibujar gráficas 2D

centroid.py > ...

```python
import pandas as pd
import matplotlib.pyplot as plt
import arff
# Leer el archivo .arff y cargarlo en un DataFrame
def load_arff(file_path):
    with open(file_path, 'r') as f:
        arff_data = arff.load(f)
    return pd.DataFrame(arff_data['data'], columns=[attr[0] for attr in arff_data['attributes']])

#file_path = 'processtimeQueueTime.arff'
file_path = '3clasesManhatten.arff'

# Cargar datos
data = load_arff(file_path)

centroids = {
    'cluster0': [0.0001, 0.0003],
    'cluster1': [0.0001, 0.0148],
    'cluster2': [0.0001, 0.0015]
}

# Nombres de los atributos
x_attr = 'sendtime'
y_attr = 'queuetime'

# Colores para cada cluster
# Colores para cada cluster
colors = {'cluster0': 'green', 'cluster1': 'blue', 'cluster2': 'red' }
colors2 = {'cluster0': 'blue', 'cluster1': 'red', 'cluster2': 'green' }

# Crear la figura
plt.figure(figsize=(10, 6))
for cluster in centroids.keys():
    cluster_data = data[data['Cluster'] == cluster]
    plt.scatter(cluster_data[x_attr], cluster_data[y_attr], c=colors[cluster], label=f'{cluster}')

# Graficar cada centroid
for cluster, (x, y) in centroids.items():
    plt.scatter(x, y, label=f'Centroide {cluster}', s=200, marker='x', c=colors2[cluster], linewidth=2)
plt.xlabel(x_attr)
plt.ylabel(y_attr)
plt.title(f' {x_attr} vs {y_attr}')
plt.legend()
plt.grid(True)

plt.savefig('Manhatten3-QT-ST.png')

```

Código para dibujar gráficas 3D

centroid.py > ...

```python
import pandas as pd
import matplotlib.pyplot as plt
import arff
# Leer el archivo .arff y cargarlo en un DataFrame
def load_arff(file_path):
    with open(file_path, 'r') as f:
        arff_data = arff.load(f)
    return pd.DataFrame(arff_data['data'], columns=[attr[0] for attr in arff_data['attributes']])

#file_path = 'processtimeQueueTime.arff'
file_path = '3clasesManhatten.arff'

# Cargar datos
data = load_arff(file_path)

centroids = {
    'cluster0': [0.0001, 0.0003],
    'cluster1': [0.0001, 0.0148],
    'cluster2': [0.0001, 0.0015]
}

# Nombres de los atributos
x_attr = 'sendtime'
y_attr = 'queuetime'

# Colores para cada cluster
# Colores para cada cluster
colors = {'cluster0': 'green', 'cluster1': 'blue', 'cluster2': 'red' }
colors2 = {'cluster0': 'blue', 'cluster1': 'red', 'cluster2': 'green' }

# Crear la figura
plt.figure(figsize=(10, 6))
for cluster in centroids.keys():
    cluster_data = data[data['Cluster'] == cluster]
    plt.scatter(cluster_data[x_attr], cluster_data[y_attr], c=colors[cluster], label=f'{cluster}')

# Graficar cada centroid
for cluster, (x, y) in centroids.items():
    plt.scatter(x, y, label=f'Centroide {cluster}', s=200, marker='x', c=colors2[cluster], linewidth=2)
plt.xlabel(x_attr)
plt.ylabel(y_attr)
plt.title(f' {x_attr} vs {y_attr}')
plt.legend()
plt.grid(True)

plt.savefig('Manhatten3-QT-ST.png')

```