Procesamiento paralelo de retrasos en los recorridos de las lineas del sistema de trasporte Metropolitano para la ciudad de Montevideo

Gabriel Acosta y Emiliano Gonzalez
Facultad de Ingeniería, Universidad de la República
E-mail {gabriel.acosta.soria, gonzalez.emiliano}@fing.edu.uy

***Resumen*—El presente artículo trata sobre el análisis de los retrasos en los horarios de las líneas de autobuses, del sistema de transporte en la ciudad de Montevideo, durante el mes de junio del año 2024. Se propone entregar una serie de indicadores sobre la demora en los servicios y la cantidad de personas que afecta este atraso. Como es un gran volumen de datos a procesar, se utilizan técnicas de programación paralela y se utilizarán para ello, las maquinas existentes en las salas de computación de la Facultad de Ingeniería.**

I. Descripción del problema

A partir de los principios para el manejo de los datos abiertos en el gobierno, [1] [2] la intendencia de Montevideo (IM) pone a disposición de la ciudadanía la mayor parte de los datos públicos que ésta maneja, para que puedan ser procesados por cualquier ciudadano, investigador, periodista, universidades y organizaciones nacionales y extranjeras [3]. Estos datos figuran tanto en la página de la IM [4], como en el catálogo de datos abiertos del Gobierno uruguayo [5].

A partir de estos mismos datos, la IM publica datos estadísticos sobre el desarrollo de ciertas áreas de interés [6], donde la movilidad en todos su aspectos ha sido relevante el los últimos años [7]. En particular, con la inclusión del sistema de transporte público metropolitano STM y su consiguiente digitalización y utilización de tarjetas de tipo MIFARE [8], sumado a la utilización de GPS para el posicionamiento de los vehículos, han ofrecido muchas posibilidades de procesamiento posterior a partir de las ventas y el seguimiento de los autobuses en su recorrido por la ciudad.

A partir de los datos abiertos publicados y de la captura de datos mediante la API propuesta por la IM [9], se propone calcular el retraso de los buses del sistema STM en Montevideo durante el mes de junio de 2024, luego relacionar estos datos con los datos de la ventas de boletos en junio del mismo año, con el fin de lograr inferir a qué cantidad de personas afecta ese retraso. Otras posibles métricas a mostrar podrían ser las paradas con más atraso, la distribución de los atrasos en el correr del día o en el mes, entre otras.

Realizamos a partir de la API de la IM, una captura cada 20 segundos de la ubicación de los buses en tiempo real en todo el mes de junio de 2024. A su vez, capturamos diariamente los datos de los horarios estimados de pasada, por cada parada para cada variante y por tipo de día (hábil, sábado y domingo). Esto lo hacemos por cualquier cambio eventual en los horarios al correr del mes.

Además, vamos a utilizar los datos de ventas de boletos en el mes de junio del mismo año. Con estos datos y el procesamiento de retrasos de buses, podemos saber cuántas personas por parada por día son afectadas por este retraso.

II. Implementación de la Solución

II-A. Justificación de usar HPC

Realizamos una captura cada 20 segundos durante 30 días. Esto nos da un total de 129600 capturas en total y un promedio de unos 400 buses por captura (máximos de 1500 buses mínimos de 50). Por lo tanto, se necesita procesar unos 52 millones de registros aproximadamente para el mes de junio. En estos registros figuran la variante de la línea, la hora de salida de la terminal (llamado por la IM frecuencia), el número de coche, la ubicación geográfica entre otros datos relevantes. Además, descargamos diariamente los archivos de horarios que se encuentra en la web de datos abiertos de la IM, por lo que contamos con 30 archivos de horarios distintos (uno por cada día del mes de junio del 2024). Es de hacer notar, que cada archivo de horarios cuenta con unas 2.5 millones de líneas. Recorriendo los datos de capturas y horarios, se pretende calcular el atraso con la hora de pasada prevista para cada parada. Encontrar la hora de pasada por cada parada, para determinada variante frecuencia y día en particular, implica un esfuerzo computacional considerable, lo que lleva a un tiempo prolongado de ejecución, si este se realiza de forma secuencial. Otro detalle adicional a considerar, es que la ubicación geográfica de las capturas de GPS se encuentra en formato WGS84 (EPSG:4326), mientras que los datos de los horarios junto con las ubicaciones de las paradas se encuentra en formato UTM 21S (EPSG:32721). Esta transformación necesaria, agrega más tiempo al procesamiento.

II-B. Pre-procesamiento de los datos

Los archivos que la IM brinda para los horarios por paradas no están geo-referenciados. Para no aumentar el cómputo, se agregan a estos archivos, los campos de las coordenadas que brinda la IM para cada parada que figura en los datos abiertos de la IM. [11]

II-C. Estrategia de resolución

Para el análisis utilizaremos estos dos términos:

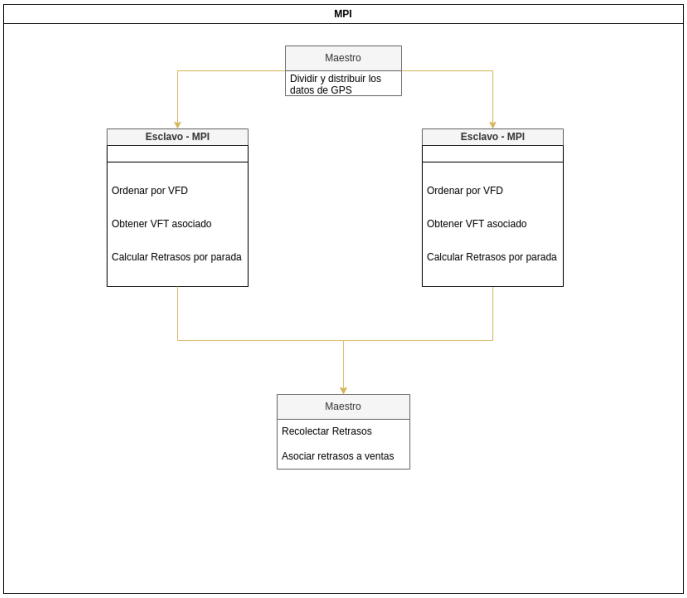

Figura 1. Esquema de resolución

- VFD (Variante, frecuencia, día):
  Representa de forma única, una variante de bus, una hora de partida desde el origen (llamada por la IM frecuencia) y una fecha determinada. Por ejemplo: variante #4735 (191 - Punta Carretas), 19020 (19:02 horas), 01/06/2024, corresponde al VFD 4735-19020-01/06/2024, estos datos se procesarán desde el archivo de capturas.
- VFT (Variante, frecuencia, tipo de día):
  Representa de forma única una variante de bus, una hora de partida desde el origen, y un tipo de día (hábil=1, sábado=2, domingo=3). Por ejemplo: 4735-19020-1, este dato se procesará desde el archivo de horarios.

En el proceso de capturas de ubicaciones de buses, generamos un archivo CSV con las capturas en el rango de una hora. Deberíamos contar con un total de 720 archivos (24 horas por 30 días). Sin embargo, debido a caídas en el sistema que creamos para la captura de datos, sumados a fallas en la API de la IM, se capturaron 702 archivos completos y 3 archivos parciales, resultando en una falta de 5 archivos de capturas lo que resulta en un 98 % de capturas aproximadamente.

El esquema de la figura 1 retrata la estrategia de resolución. Para la implementación se utilizó un modelo maestro/esclavo usando un esquema de descomposición de dominio. Esta implementación se realizo en C con la biblioteca MPI.

Los datos se agrupan en directorios,para las capturas es uno por cada día (al cual le llamamos bloque), por su parte los horarios se guardaran en un directorio, donde el nombre denota el día al que pertenece. Cada proceso procesara al menos un bloque, con su correspondiente horario. El proceso master divide de forma equitativa, la ruta de los directorios entre los procesos esclavos, utilizando para esto MPI Scatter. Tanto el master como los esclavos primero agrupan las capturas de sus bloques asignados por VFD, y para cada VFD consiguen los datos del VFT , desde el archivo de horarios correspondiente, generando archivos temporales que luego serán utilizado por el algoritmo caja negra. Este algoritmo consta de un script en Python que utiliza bibliotecas de PyQGIS para el calculo de atrasos. Este algoritmo, retorna los atrasos correspondientes para cada parada en las que existen capturas cercanas y luego elimina todo tipo de capa o memoria que pudo haber utilizado el QGIS. Al final del procesamiento de cada bloque, los esclavos retornan sus resultados al master, pero antes realizan un pre-procesamiento para sacar datos incongruentes que surgen a partir del calculo de los atrasos.

Una vez procesados los atrasos de todo el mes, el proceso master sumariza los retrasos y los cruza con los datos de las ventas de boletos del mismo mes,para cada variante, parada y día, con el fin de calcular la cantidad de personas afectadas en cada parada por los retrasos en las lineas. Inicialmente consideramos procesar los archivos de ventas de Junio de años anteriores, debido al atraso en la publicación de los datos. Afortunadamente, pudimos contar con las ventas de pasajes en el mes de Junio del presente año. Con estos números, se tiene una idea de a que cantidad gente afecta los atrasos.

Los datos obtenidos desde la API se dividieron por bloques de 24 horas. Estos bloques son distribuidos dependiendo de la cantidad de procesos con los que se cuente, pero para poder controlar el correcto funcionamiento de los nodos, con las pruebas realizadas, se determino que, la cantidad de bloques (días) a distribuir sera de una unidad. Con las medidas realizadas, se llega a que la contabilizacion de las 24 horas de un día, insume aproximadamente media hora.

Cada proceso deberá ordenar los datos por VFD antes de procesarlos. Esto hará que cada uno de ellos cuente con datos totales o parciales para determinado VFD. Luego del ordenamiento se realizará el procesamiento de los datos.

Inicialmente, se pensó en separar por bloques desde las 3 AM (aproximadamente), los datos serian divididos entre bloques de 24 horas hasta las 2 AM del día siguiente, pero debido al poco tiempo en que realizamos las pruebas, optamos por separar en bloques (carpetas) con las 24 horas del día. De igual manera, los horarios de los buses cambian todos los días, así que generaremos un archivo para cada día. Como el archivo de Python procesara todo un día, al final del procesamiento hecho por el código en C, los VFD parciales serán procesados por el master una vez reciba los datos de los esclavos.

Se implementaron dos algoritmos: uno para procesar el conjunto de datos en cada proceso (ordenar en VFDs y obtener VFTs), esta implementación se realiza en C y luego se utiliza para el cálculo de la demora, un script de Python utilizando librerías de QGIS [10], que realiza las siguientes tareas: transformar las coordenadas de las paradas del VFT de los sistemas WGS84 a UTM 21S, determinar qué paradas del VFT serán cubiertas por el recorrido del VFD (ya que pueden ser parciales). Luego, para estas paradas válidas se calcula la hora de pasada utilizando para ellos los dos registros más cercanos, retornando así la demora en minutos para cada parada e incluyendo datos relevantes para un posterior análisis.

II-D. Post-procesamiento de los resultados

Una vez el nodo termina de analizar el bloque, la información la procesa y se la envía al master. Este a su vez, fusiona

los datos para devolver la demora para cada parada para cada día, variante y enlaza estos datos con la cantidad de vetas por parada, para de esta forma estimar como afecta este retraso.

Es importante hacer notar que todo este mecanismo aplica cuando la aplicación se corre con mas de un proceso. En el caso de ejecutar con un único proceso, su ejecución sera secuencial.

III. Análisis experimental

III-A. Escalabilidad

- Comunicación entre procesos:
  La comunicación entre procesos se realiza al estilo rendevous, donde cada esclavo envía sus resultados calculados para cada día, utilizando un barrier para sincronizar los procesos y luego realizando envíos bloqueantes al master.
- Balance de carga:
  Se utiliza una arquitectura de maestro-esclavo, donde el maestro distribuye los bloques de forma equitativa entre los esclavos utilizando MPI Scatter.
- Granularidad:
  La granularidad esta definida por día de capturas y horarios, con el fin de minimizar los cálculos parciales de atrasos en el recorrido y evitar posibles números erróneos.

III-B. Rendimiento

Para evaluar el rendimiento del sistema se calcularán diferentes métricas:

- Tiempo de ejecución en minutos (TE).
- Speedup logarítmico.
- Eficiencia.

Como ya se menciono anteriormente, en el caso de ejecución con un único proceso, esta se realiza de forma secuencial lo que implica que las métricas se basen en calculo de Speedup algorítmico y no de paralelidad.

III-C. Evaluación experimental

Cada una de estas métricas se calcularon sobre diversas cantidades de procesos. En la siguiente tabla de la figura 2 se observan los resultados de los datos:

Nodos	Procesos	TE	Speedup	Eficiencia
1	1	293	1	1
2	2	276	1,06	0,53
3	3	261	1,12	0,37
5	5	228	1,29	0,26
6	6	214	1,36	0,23
15	15	170	1,72	0,11
30	30	104	2,8	0,09

Figura 2. Tabla de resultado de los datos

En el gráfico de la figura 3, podemos observar los valores de speedup calculados versus la función lineal:

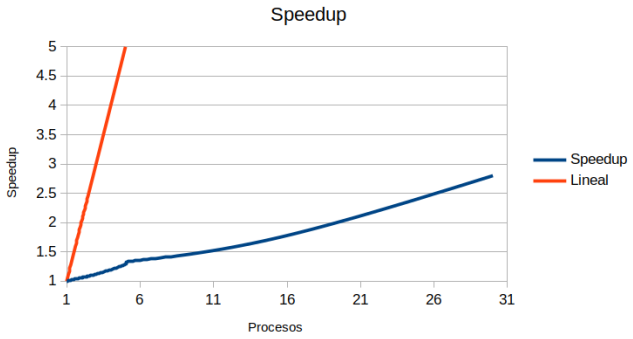

Figura 3. Gráfica del speedup

III-D. Discusión de resultados experimentales

Notamos que corremos un proceso por nodo debido a que el algoritmo de Python que corre QGIS consume una gran cantidad de recursos de memoria (memory-bound). Además, la cantidad de procesos/nodos es divisor de la cantidad de bloques asignados debido a la limitación del MPI Scatter convencional. Otro detalle a mencionar es que, dado que la granularidad esta dada a nivel de bloques, correr la aplicación con mas de 30 procesos seria contraproducente debido que los procesos quedarían ociosos ya que no tendrían tareas a realizar.

Mediante los cálculos de tiempo estimado, speedup y eficiencia notamos que el rendimiento del sistema no mejora de manera deseada a medida que aumenta la cantidad de procesos. Creemos que esto sucede en parte por estas razones:

- La imposibilidad de correr mas de un proceso por nodo debido al gran consumo de memoria del algoritmo en PyQGIS.
- El overhead de comunicación entre los esclavos y el master para recolectar los retrasos.
- Cálculos extra realizados por el proceso master que se podrían haber paralelizado como por ejemplo la asociación de retrasos con ventas del mismo mes, debido a que lamentablemente por falta de tiempo, no pudimos implementarlo de forma paralela.

III-E. Posibles mejoras a realizar

Dentro de las posibles mejoras a realizar para mejorar la performance del sistema, se encuentran:

- Disminuir la cantidad de datos con la cual se ejecuta el algoritmo PyQGIS con el fin de disminuir el consumo de memoria y poder ejecutar mas de un proceso por nodo, aprovechando las ventajas de la localidad.
- Paralelizar en lo posible, tareas existentes que se realizan de forma secuencial.
- Utilización de ventanas de memoria compartida entre procesos. Esto se intento realizar de forma fallida.
- Utilizar MPI Gather para recolectar los resultados de los esclavos en vez de sincronización y comunicación bloqueantes del estilo rendevous.

IV. Resultados obtenidos y Conclusiones

IV-A. Resultados del procesamiento

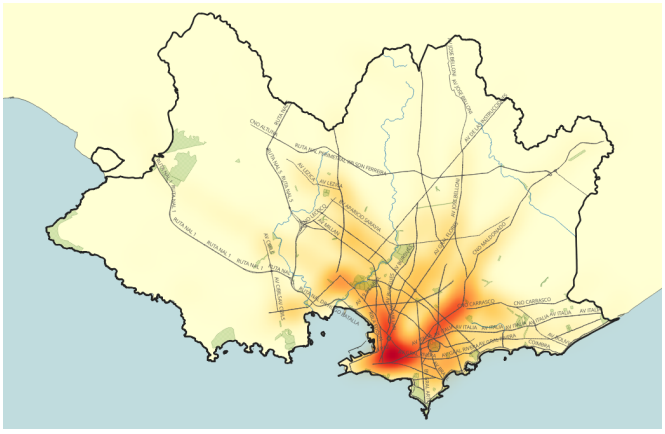

Figura 4. Mapa de calor con los atrasos

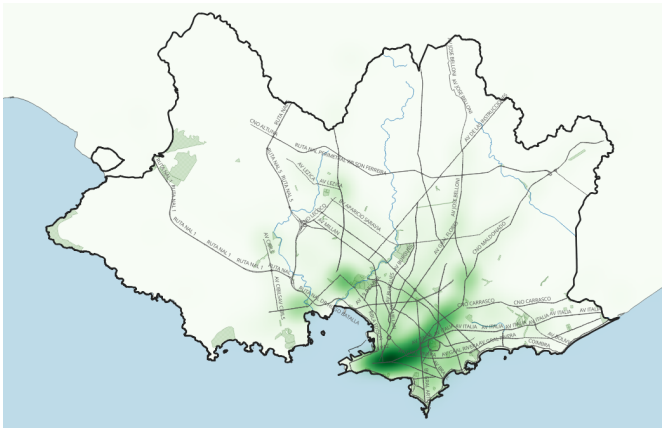

Figura 5. Mapa de calor con los pasajeros afectados por los retrasos

Si observamos los mapas de calor de retrasos por parada de ómnibus (Figura 4) y venta de boletos en esas paradas (Figura 5), se puede ver que los focos de mayor retraso coinciden con mayores ventas.

Como muestra la Figura 6, los barrios mas afectados por los atrasos de los ómnibus son Centro, Cordón, La Unión, Aguada, La Blanqueada y Belvedere.
Y las avenidas con mas atrasos son 18 de Julio, 8 de Octubre, Agraciada y Carlos María Ramírez.

En la figura 7 se muestra , para las personas afectadas por los retrasos, que porcentaje de estas se distribuidas por grupos de atrasos. Observando la gráfica, podemos notar que la mayoría de retrasos son de menos de 5 minutos. Sin embargo, existe una cantidad considerable de retrasos de hasta 10 minutos.

IV-B. Conclusiones

En este informe, hemos mostrado la efectividad del uso de técnicas de programación paralela para procesar grandes

Figura 6. Mapa de calor con focos de los atrasos

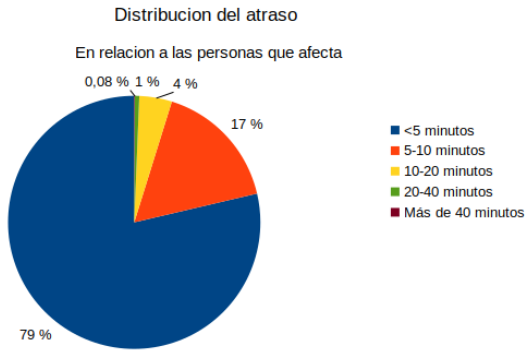

Figura 7. Gráfico que muestra el porcentaje de personas afectadas por determinados minutos de retraso

volúmenes de datos relacionados con los retrasos en el sistema de transporte metropolitano de Montevideo. La implementación de un enfoque distribuido en los sistemas de cómputo disponibles en la Facultad de Ingeniería permitió procesar los datos de manera más eficiente, reduciendo de cierta manera el tiempo de cálculo en comparación con enfoques secuenciales.

Los resultados obtenidos revelan patrones importantes en los retrasos de las líneas de buses, identificando tanto las zonas con atraso,como las corredores más afectadas. Estos hallazgos podrían ser útiles para la Intendencia de Montevideo y las empresas de transporte publico en la planificación de mejoras, ya sea en la red , en los horarios, y las ubicaciones de las paradas, para descongestionar el uso intensivo de estas y para con ello optimizar la puntualidad del servicio.

Referencias

[1] Principios para el manejo de datos abiertos en el gobierno — Intendencia de Montevideo.
URL: https://montevideo.gub.uy/areas-tematicas/servicios-digitales/datos-abiertos/principios-para-el-manejo-de-datos-abiertos-en-el-gobierno
[2] 8 Principles of Open Government Data.
URL: https://public.resource.org/8_principles.html
[3] Servicios digitales — Intendencia de Montevideo.
URL: https://montevideo.gub.uy/areas-tematicas/servicios-digitales
[4] Portal de Datos Abiertos — Intendencia de Montevideo.
URL: https://ckan.montevideo.gub.uy/
[5] Catálogo de Datos Abiertos — GUB.UY
URL: https://catalogodatos.gub.uy/
[6] Datos Abiertos Ambientales — Intendencia de Montevideo.
URL: https://montevidata.montevideo.gub.uy/
[7] Movilidad — Intendencia de Montevideo.
URL: https://montevidata.montevideo.gub.uy/movilidad

[8] MIFARE: Contactless Chips.
URL: https://es.wikipedia.org/wiki/Mifare
[9] API de Transporte Publico — Intendencia de Montevideo.
URL: https://api.montevideo.gub.uy/apidocs/publictransport
[10] QGIS — Sistema de Información Geográfica libre y de Código Abierto.
URL: https://www.qgis.org/es/site/about/index.html
[11] Transporte colectivo: paradas, puntos de control y recorridos de omnibus. URL: https://catalogodatos.gub.uy/dataset/transporte-colectivo-paradas-puntos-de-control-y-recorridos-de-omnibus