Análisis de ventas de repelentes con relación a las invasiones de mosquitos de 2024

Agustín Recoba
Universidad de la República
Facultad de Ingeniería
Montevideo, Uruguay
agustin.recoba@outlook.com

Agustín Martinez
Universidad de la República
Facultad de Ingeniería
Montevideo, Uruguay
amartinez@fing.edu.uy

***Abstract*—En febrero y marzo de 2024, una invasión de mosquitos en la región del Río de la Plata afectó principalmente a Argentina y Uruguay. Este fenómeno provocó un aumento significativo en las ventas de insecticidas y repelentes.**

Utilizando datos de ventas proporcionados por Scanntech UY, aplicamos técnicas de computación de alta performance para analizar tendencias de ventas y comparar con otros productos. Este estudio busca confirmar los reportes mediáticos y medir el efecto de las publicaciones en las tendencias de compra en Uruguay.

Este documento sirve como informe del proyecto de fin de curso de HPC o Computación de Alta Performance (edición 2024).

***Index Terms*—HPC, clúster, Hadoop, ventas, mosquitos, invasión, Uruguay**

I. Introducción

En el verano de 2024, en particular febrero y marzo, una invasión de mosquitos en la región del Río de la Plata generó un aumento notable en la demanda de productos como insecticidas y repelentes. En Argentina, la sobre demanda provocó desabastecimiento masivo en las tiendas minoristas [8].

Si bien en Uruguay también se manifestó un fenómeno similar, donde hubo un aumento sustantivo de las ventas de repelentes e insecticidas, este fue de menor magnitud que en el país vecino [13].

En este trabajo se utilizan datos de ventas de minoristas de Uruguay, proporcionados por Scanntech UY, para analizar las tendencias de precio/venta durante y después de la invasión de mosquitos, utilizando técnicas de computación de alta performance (HPC) basadas en Apache Hadoop [3] a partir de jobs MapReduce (ver Sección I-A). En particular, lo que interesa a este trabajo es investigar los cambios de tendencia de precio y cantidades de ventas por día y categoría, para determinar si existieron fenómenos como desabastecimiento.

La invasión afectó principalmente al sur del país, en particular al Área Metropolitana de Montevideo, a raíz de esto, filtramos las tuplas que no correspondían a ventas de Montevideo y Canelones.

Una fecha clave para este trabajo es la llegada de la invasión de mosquitos a Uruguay y su consecuente anuncio en prensa [14] [15] y a través de medios oficiales [11]: el 23 de febrero de 2024, donde esperamos que, a partir de ese momento, se observen cambios en la tendencia de la venta de repelente e insecticidas.

A. Introducción a Hadoop MapReduce

En un job MapReduce, el proceso se divide en varias fases que se ejecutan tanto en paralelo como secuencialmente para procesar grandes volúmenes de datos de manera eficiente. La primera fase es la fase de mapeo (Map phase), donde los datos de entrada se dividen en fragmentos y cada fragmento se procesa en paralelo por tareas de mapeo que transforman los datos en pares clave-valor intermedios. Luego, los resultados intermedios se ordenan y agrupan por claves en la fase de barajado (Shuffle phase), que también se realiza en paralelo y puede comenzar antes de que todas las tareas de mapeo hayan terminado, optimizando así el tiempo de procesamiento total. En la fase de reducción (Reduce phase), los pares clave-valor agrupados se procesan por tareas de reducción, que combinan los valores asociados a cada clave para producir los resultados finales. Aunque las fases de mapeo y reducción se ejecutan en paralelo en múltiples nodos para maximizar la eficiencia, la fase de barajado es necesaria para reorganizar los datos y permitir que las tareas de reducción procesen las claves adecuadas, introduciendo una dependencia secuencial entre las fases de mapeo y reducción.

El modelo MapReduce es una de las partes de la estructura básica de Hadoop, un framework para programar aplicaciones distribuidas que manejen grandes volúmenes de datos. Los otros tres módulos principales que conforman el ecosistema Hadoop son:

- *Hadoop Distributed File System* (HDFS): Es el sistema de archivos distribuidos de Hadoop, con características como alta tolerancia a fallos, escalabilidad y rendimiento en el acceso secuencial de datos. En HDFS los datos se dividen en bloques y se replican en múltiples nodos para asegurar disponibilidad y permitir procesamiento paralelo.
- *Yet Another Resource Negotitatior* (YARN): Es el sistema de gestión de recursos de Hadoop, permite la ejecución de aplicaciones distribuidas en un clúster. Separa las funciones de gestión de recursos y programación de tareas. Consta de un ResourceManager, que gestiona la asignación de recursos, y NodeManagers, que monitorean los recursos y la ejecución de tareas en cada nodo.

- Bibliotecas: Son herramientas utilizadas y compartidas por otros módulos de Hadoop.

Lo que se mencionó anteriormente sobre la tolerancia a fallos del HDFS se extiende en general a todo el framework: *Apache Hadoop continúa ejecutándose incluso si fallan uno o varios nodos de procesamiento de datos. Realiza varias copias del mismo bloque de datos y las almacena en distintos nodos. Cuando un nodo falla, Hadoop recupera la información de otro nodo y la prepara para el procesamiento de datos.* [2]. Gracias a esta característica, podemos despreocuparnos por implementar explícitamente la tolerancia a fallos. Entonces, el foco de la etapa de diseño e implementación de este proyecto estará en conceptualizar y desarrollar un pipeline utilizando la estrategia MapReduce que resuelvan el problema planteado. En secciones posteriores se desarrolla en profundidad sobre la solución utilizando cadenas de jobs MapReduce.

II. Descripción de los datos obtenidos

Los datos cedidos por Scanntech UY son tres tablas en formato Comma Separated Value (csv): $VENTAS$, $REL\_LOCALES$ y $REL\_PRODUCTOS$. En esta sección describiremos brevemente el contenido de cada una, sus campos y el motivo de uso. Además, para combinar esta información, es necesario realizar uno o más joins, esto también se implementará como jobs MapReduce, para aprovechar la capacidad de procesamiento paralelo.

A. Ventas de locales minoristas

Datos de ventas, distribuido en varios archivos que contienen los atributos: *código de local, código de producto, fecha de la venta, número de ticket, cantidad de unidades* y *precio por unidad*. Cada línea se interpreta de la misma forma que se interpreta una línea en una factura o "ticket" que recibe uno en un supermercado o almacén. Los registros con el mismo código de local, código de ticket y fecha corresponden a una misma compra o "pago".

B. Relación de locales

Tabla de locales, asociando el código de local mencionado con los datos del mismo, como departamento donde está ubicado. Dado que el tamaño es razonablemente pequeño, un solo archivo las contiene. Obtener los locales fue necesario para acotar la búsqueda a los Departamentos de Montevideo y Canelones.

C. Relación de productos

Tabla de productos, asociando el código de producto con datos del mismo, como puede ser código de barras, cantidad de contenido o categoría. Para esto también se usa un solo .csv.

III. Metodología

Para analizar los datos de ventas de aproximadamente 60 GB, correspondientes a los meses de abril, marzo y mayo de 2023 y 2024, utilizamos un clúster Apache Hadoop [3]. La solución se modeló a partir de jobs MapReduce implementados en Java. El repositorio de código generado se adjunta con la entrega en la carpeta 'mosquito'.

A. Plataforma y desarrollo

Se configuró un clúster Hadoop en la infraestructura Cloud de Google [5] para hacer todo el procesamiento pesado. Para algunas tareas más livianas usamos un clúster local de un único nodo basado en contenedores docker [6]. Este mismo tipo de configuración se usó para las pruebas durante el desarrollo del proyecto [4].

Además, adjuntaremos a la entrega el proyecto Maven Java con el código completo.

¿Por qué usamos DataProc? [5] Solicitamos acceso al ClusterUY y planificamos desde un inicio usar esa plataforma para el procesamiento pesado, pero por cómo se gestiona el acceso a los nodos fue imposible hacer andar Hadoop. El principal inconveniente es que no permite hacer SSH a nodos si no es a través del uso de sesiones interactivas, lo que no es compatible con el sistema de comunicación base de Apache Hadoop. Por otro lado, también descartamos el uso de las máquinas de la facultad (las *pcunix*) porque era imposible que la URI nos permita acceder a aprox. 300 GB de espacio en disco. Las únicas soluciones que quedaban eran armar un mini clúster local o usar un IaaS. Explorando las opciones encontramos esta solución y haciendo uso de USD 300 de prueba que ofrece Google, decidimos que era la mejor opción.

B. Obtención de Datos

Los datos de ventas fueron recolectados por Scanntech UY y abarcan un amplio rango de minoristas en Uruguay. Los datos incluyen transacciones de productos categorizados como insecticidas y repelentes, así como otros productos para control y comparación (refrescos).

C. Técnicas de Análisis

Los datos se analizaron utilizando cadenas de jobs MapReduce en el clúster Hadoop. Se aplicaron algoritmos de Regresión Segmentada y variaciones de PELT para detectar tendencias en las series temporales de ventas.

D. Técnicas de JOIN entre las tablas

Como adelantamos anteriormente, para combinar información de ventas con la información de productos y locales, es necesario joinear dichas tablas, esto se hizo de la siguiente manera:

$$\prod_{\substack{categoria \\ fecha\_venta \\ cant\_venta \\ departamento}} (PRODUCTOS) * (REL\_VENTAS) * (REL\_LOCALES)$$

Para esto, vamos a intentar aprovechar la diferencia de tamaño entre las tablas para minimizar los costos de comunicación.

Dentro de este tipo de operaciones, elegimos probar dos opciones. El principal candidato es Broadcast Join, es un join que se realiza en la etapa de mapping y que requiere que las tablas chicas se hayan guardado en un caché distribuido

y esté accesible de forma completa para cada Mapper. Tiene la ventaja de que permite concatenar otros Mapper luego del Join, como por ejemplo para aplicar un filtrado, todo previo al Shuffle y Sort, reduciendo significativamente el costo de comunicaciones asociado a esta etapa. Otra ventaja importante es que el broadcast join es lo que se denomina multi-way, o sea, que permite joinear varias tablas a la vez.

La otra opción fue un Reducer-side join, algoritmos MapReduce que usan ambas etapas para obtener el resultado. Este Join no es multi-way si la clave no es la misma para todas las tablas a joinear, por lo que cada clave requiere de una fase completa MapReduce. La estrategia para este join es mapear los dos tipos de archivos a joinear a la clave por la que se joineará y en el valor agregar a qué tabla pertenece el registro actual, además del registro en sí. Por ejemplo, si queremos hacer join de ventas con productos, el mapper de ventas genera registros (codigoProducto, #VENTA [RESTO DE LA LINEA]) y el mapper de productos genera registros (codigoProducto, #PRODUCTO [RESTO DE LA LINEA]). De esta forma, el reducer luego obtiene todos los registros con el mismo código de producto, concluyendo el join.

Se descartaron tipos de join más sofisticados y eficientes, pues requieren condiciones que no somos capaces de cumplir sin un preprocesamiento costoso de los datos. Ejemplo de esto puede ser Merge Join, que requiere que las tablas estén previamente ordenadas por la clave que se usará en el join.

IV. Detección de tendencias: adaptación de algoritmos tradicionales a jobs MapReduce

El objetivo de esta parte, y en general de todo el proyecto, fue encontrar **cambios de tendencia en el precio** de las categorías muestreadas, esto se justifica en que el aumento de precio es uno de los parámetros observables en un fenómeno de desabastecimiento.

Este creemos que fue el mayor desafío del proyecto. Consistió en convertir algoritmos secuenciales, en una concatenación de jobs MapReduce, manteniendo la correctitud y haciéndolo de forma eficiente.

A. Diseño global de la solución

Dada la amplia variedad de jobs a implementar para experimentar con todas las posibles soluciones mencionadas, desarrollamos de base un mini-framework para acelerar el desarrollo de la solución en sí. Esto incluye clases para manejar Keys MapReduce compuestas, valores multivaluados, un Driver genérico para orquestar la corrida de varios jobs y varios reducers sencillos para operaciones simples como sumar, contar, etc. Esto se puede encontrar en el repositorio en las siguientes clases:

- **GroupByReducer**
  - SumAll
  - ValueCount
  - Average
- **MultiJobDriver**: Driver genérico y parametrizable para ejecutar muchos jobs MapReduce de forma consecutiva, cada uno tomando como entrada la salida del anterior.
- **Parsers**
  - Productos: para leer el archivo de productos obtenido.
  - Locales: ídem.
  - Ventas: ídem.
- **ChangePointDetectionExtensible**
  - CPAlgorithmIface: Interfaz para implementar algoritmos de detección de puntos de quiebre sobre un conjunto de puntos (Fecha, Float)
  - CPReducer: Reducer que ejecuta un CPAlgorithmIface, parametrizable.
  - CPDriver: Driver que ejecuta un job de dos fases MapReduce: Detección de puntos de cambio por producto + agregación de resultados a nivel categoría.
- **TextArrayWrittable**
- **CacheHdfs**: Maper, Reducer y Drivers programados para tener acceso a archivos del caché distribuido de Hadoop.
- Otros útiles:
  - IdentityMapper
  - CustomKey
  - ValuePair
  - MaxFloatReducer

Ninguno de estos módulos son parte de la solución persé, pero son la base de la misma, o ayudaron a lo largo del proyecto para obtener resultados intermedios, probar ideas de algoritmos o estandarizar patrones que se repetían.

B. Granularidad del cómputo y consideraciones sobre los datos

1) Granularidad mínima: El objetivo es poder hacer conclusiones AÑO NORMAL vs. AÑO CON INVASIÓN y CATEGORÍA MOSQUITOS vs. OTRA CATEGORÍA. Llevándolo al ámbito MapReduce, nos dice que nuestra clave final debería incluir esos campos.

2) Precios dispares dentro de una misma categoría: Como tenemos una base de miles de productos distintos, cada uno con un precio base distinto, debemos tener cuidado de que el algoritmo de detección de tendencia no confunda una subida de precio con la venta de un producto que simplemente es más caro. Para solucionar esto, dichos algoritmos correrán con un subconjunto de datos correspondientes a un solo producto por vez. Dicho de otro modo, el código de producto será parte de la clave siempre que se esté ejecutando un algoritmo de detección de puntos de cambio.

C. Regresión Segmentada [12]

La regresión segmentada se realiza particionando lo que se entiende como la variable independiente en regiones (varias dimensiones) o segmentos (caso unidimensional). Para cada una de estas regiones se realiza una regresión sobre la variable dependiente, obteniendo de esta forma varias aproximaciones para cada segmento.

Basándose en cierto umbral (hiperparámetro a definir) α, se contrastan las aproximaciones hechas sobre regiones consecutivas (caso unidimensional) o limítrofes (varias dimensiones),

para evaluar si la diferencia entre las aproximaciones tiene una significancia mayor a α. De esta manera se obtiene finalmente los llamados "breakpoints", puntos de quiebre en la tendencia subyacente a los datos. La implementación se encuentra en la clase *CoreSegementedRegression* y el parámetro que mejor funcionó fue $\alpha = 0.7$.

Esto lo implementamos con una cadena de jobs MapReduce de 2 fases:

1) Computo de tendencias y join
   a) Map compuesto: Broadcast Join entre las 3 tablas, filtrado de categorías que nos interesan y partición de dominio básica.
   b) Reduce: Ejecución de Regresión Segmentada en cada partición.
2) Extracción de resultados a nivel categoría
   a) Map: Agrupar productos por categoría.
   b) Reduce: Combinar los puntos de cambio detectados en todos los productos de la categoría.
   c) Reduce: Combinar los puntos de cambio detectados en todos los productos de la categoría.

El código de esta parte se puede hallar en la clase *SegmentedRegressionDriver*.

D. Pruned Exact Linear Time [9]

PELT (Pruned Exact Linear Time) es un algoritmo que particiona en segmentos de tiempo usando programación dinámica sobre los posibles bordes de los segmentos. Los bordes de dichos segmentos serán en definitiva los puntos de quiebre en la tendencia. Además, para limitar el particionamiento solamente a donde es realmente necesario, se precisa del uso de una función de costo que querremos minimizar.

El desafío aquí será la correcta adaptación del método (que es inherentemente secuencial por el uso de la memorización de la programación dinámica) y la elección de una función de costo que pueda computarse eficientemente sobre los datos que tenga el mapper o reducer de forma local.

Para lograr eso nos basamos en un paper reciente de los mismos autores originales de PELT [16], donde revisan dos formas de paralelizar el algoritmo base: Chunk y Deal.

Paralelización de PELT: Chunk y Deal

Ambas técnicas se basan en la idea de correr el algoritmo base en dos fases. La primera fase particiona la variable independiente en X segmentos no disjuntos y calcula PELT base en cada uno de ellos. La segunda fase, que toma el resultado de la fase anterior, vuelve a correr PELT base, pero ahora sobre todo el espacio de puntos y restringiendo los puntos de cambio **candidatos** a aquellos hallados en la fase anterior. En el paper [16] se muestra que este algoritmo no logra minimizar la función de costo, pero sí lo **aproxima** lo suficientemente como para considerarla una estrategia válida, especialmente comparada contra otras estrategias distribuidas que no garantizan optimalidad.

Chunk

La diferencia entre Chunk y Deal es cómo se realiza la partición de X. **Chunk** particiona en L segmentos contiguos que

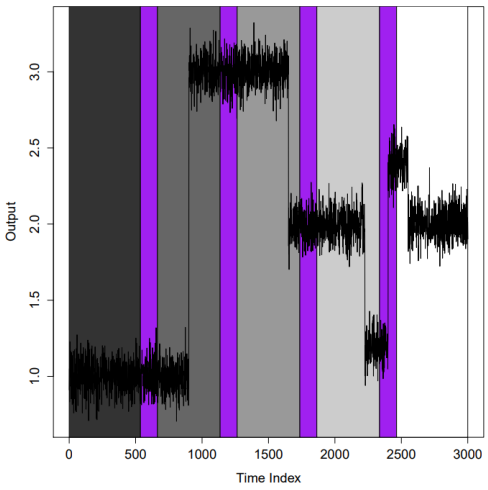

Fig. 1. Segmentación de Chunk en la primera fase. En violeta los segmentos que "se pisan".

se pisan de la siguiente forma: Sea $X = 1, ..., n$ el intervalo sobre el que queremos particionar, $L(n)$ una constante que representa la cantidad de workers disponibles para ejecutar el cómputo de cada segmento y $V(n)$ es una constante que representa cuánto overlap entre segmentos queremos (debería elegirse para balancear la precisión y rapidez). Decimos que L y V son constantes porque se deciden previo a la ejecución y el algoritmo no hace comprobaciones de correctitud sobre ellos. **Chunk** particiona en L segmentos contiguos, definiendo el segmento i-ésimo como el rango desde $(i-1) \times trunc(n/L) - V$ hasta $i \times trunc(n/L) + V$. De esta forma, cada segmento tiene un tamaño (en cuanto a índices máximos y mínimos) igual o semejante a $\frac{X}{L}$.

Deal

Deal se basa en cómo se reparten las cartas en el casino, 1 carta para cada jugador, de forma cíclica. De esta forma cada segmento tiene un tamaño (en cuanto a índices máximos y mínimos) igual o semejante a X. Aquí no hay overlap en los segmentos, conceptualmente no son necesarios porque cada segmento ya tiene cierto "conocimiento" global de la serie.

Nuestro acercamiento

Para nuestro acercamiento decidimos imitar la solución Deal, principalmente porque no registra un overlap entre los segmentos y eso en definitiva ahorra cómputo de forma global. Además, en el paper [16] no lograron establecer una relación lo suficientemente fuerte entre el overlapping de Chunk con la precisión final del algoritmo, pero sí se ve que Deal escala mejor que Chunk.

El pasaje a MapReduce lo logramos con una cadena de jobs de 3 fases:

1) Fase 1 de PELT Deal
   a) Map compuesto: Broadcast Join entre las 3 tablas, filtrado de categorías que nos interesan y agrupamiento en segmentos aproximando la estrategia PELT Deal,
   b) Reduce: Ejecución de PELT sobre cada segmento.

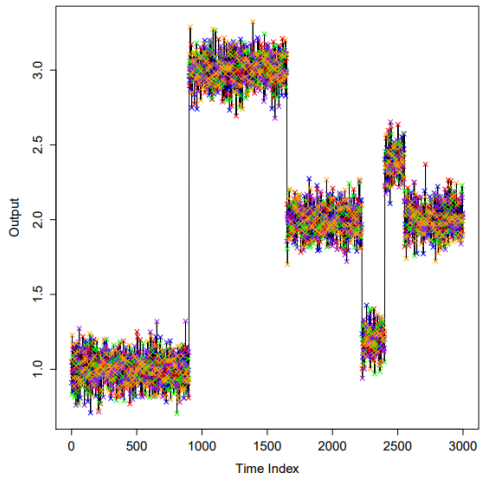

Fig. 2. Segmentación de Deal. Cada color representa un segmento distinto. No hay overlapping.

2) Fase 2 de PELT Deal
   a) Map: Agrupar segmentos anteriores.
   b) Reduce: Ejecución de PELT sobre el segmento completo pero restringiendo a los puntos de cambio hallados en la etapa anterior.
3) Extracción de resultados a nivel categoría
   a) Map: Agrupar productos por categoría.
   b) Reduce: Combinar los puntos de cambio detectados en todos los productos de la categoría.

El código de esta parte se puede hallar en la clase *DealPELTDriver*.

Además, vamos a implementar PELT base, de una sola fase, para comparar cómo se desempeña la versión paralelizada con la versión base. Consultar código en la clase *BasePELTDriver*.

V. Pruebas preliminares y elección de algoritmo final

Para decidir qué algoritmo de join usaremos en los algoritmos de detección de tendencias, hicimos una etapa de pruebas previa, independiente de este cálculo.

En esta etapa calculamos el total de venta de cada categoría, por departamento y por fecha. De esta manera conseguimos esos datos, que van a servir para entender el fenómeno de la invasión y además, logramos extraer conclusiones acerca de cuál algoritmo de join es más adecuado para este caso.

Estas dos implementaciones las encontramos en las clases *TestHdfsHashJoinSum* y *ReduceJoinDriver*. Ambas implementan joineado y filtrado de ventas, locales y productos. La primera lo logra solamente con un Map y un Reduce, mientras que la segunda hace uso de 3 de cada. Como mencionamos antes, el Reduce-Side join necesita una fase Map-Reduce completa para joinear cada tabla (aquí queremos joinear dos) y luego necesitamos una última fase para hacer la suma y obtener los valores totales.

A. Análisis de estrategias de JOIN

Para estas conclusiones corrimos los dos algoritmos en un nodo local, de 6 cores / 12 threads, con 32 GB de memoria RAM disponible. Nos limitamos a correr únicamente con ventas de tamaño 200 MB, 2 GB y 19 GB. El objetivo es predecir qué algoritmo se desempeñará mejor al ser usado en conjunto con el algoritmo de detección de cambios de tendencia sobre el dataset de 60 GB.

Broadcast Join vs Reducer-side Join

TABLE I
Resultados de BroadcastJoin

Dataset	200 MB	2 GB	19 GB
Tiempo serial	1296	1440	2520
Tiempo paralelo	180	240	630
Speedup Algoritmico	7.2	6,0	4,0

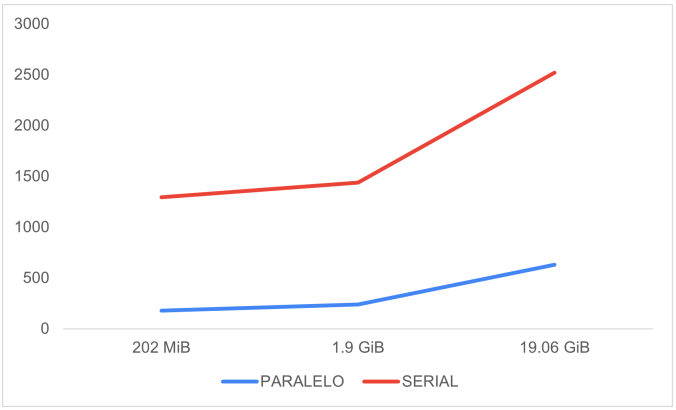

Fig. 3. Tiempos de test de performance de Broadcast Join. Serial (1 hilo) vs. paralelo (hasta 12 hilos).

TABLE II
Resultados de ReducerJoin

Dataset	200 MB	2 GB	19 GB
Tiempo serial	1134	1560	2640
Tiempo paralelo	180	246	840
Speedup Algoritmico	6.3	6.3	3.1

En vista de los resultados (Fig. 3 y4, Tab V-A y Tab. V-A), vemos que la estrategia de Broadcast Join muestra una mejor performance que el Reducer Side Join; sin embargo, la diferencia no es sustantiva, de unos pocos segundos para los datasets pequeños, y unos pocos minutos para el caso de los datasets grandes.

Observamos que el speedup algorítmico decrece cuanto más grande es el dataset. Esto se puede explicar por varios factores. A medida que el tamaño del dataset crece, las partes secuenciales del algoritmo, según la Ley de Amdahl, pueden tener un impacto mayor en el tiempo total de ejecución. Además, el overhead de comunicación y sincronización, limitaciones de memoria, entre otros, pueden incrementar con datasets más grandes, afectando negativamente el speedup.

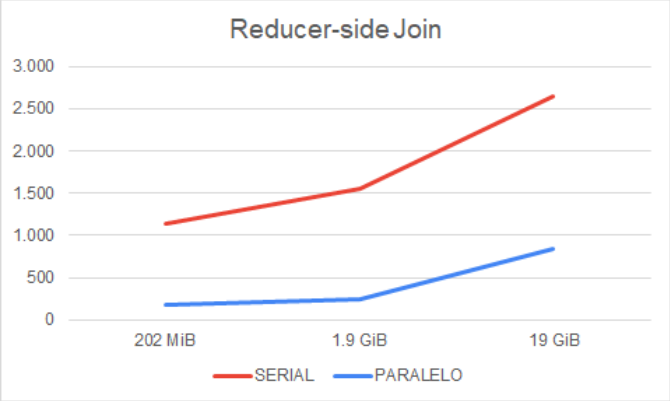

Fig. 4. Tiempos de test de performance del Reducer-side Join. Serial (1 hilo) vs. paralelo (hasta 12 hilos).

Aunque sospechamos que podría existir un overhead constante al comienzo de la ejecución que tenga más peso en pruebas cortas, este no sería el único factor que influye en la disminución del speedup con datasets más grandes.

VI. Resultados experimentales: performance de la solución

En esta sección exploraremos la paralelización de los algoritmos implementados. Nos centraremos en el uso del Speedup Algoritmico como medida de ello.

Para asegurarnos de que una corrida de Hadoop sea puramente secuencial, usamos los parámetros *mapreduce.job.running.map.limit=1* y *mapreduce.job.running.reduce.limit=1*. El uso de estos parámetros implica que la cantidad máxima de mappers y reducers (respectivamente) ejecutándose en paralelo sea siempre menor o igual a 1. Para tener el tiempo del algoritmo paralelo utilizando N recursos, lo que haremos será **no limitar la concurrencia** y **escalar el tamaño del clúster** a través de la interfaz que brinda DataProc [5] para ello.

Las especificaciones técnicas de las VM usadas para cada nodo corresponden a la categoría ”n1-standard-4” (4 CPU virtuales y 15GB de RAM), más detalles pueden consultarse en [7]. En cuanto a performance de la red usada, no notamos en ninguna de las pruebas preliminares que esto haya sido un cuello de botella para el procesamiento de los datos. Además, todos los archivos involucrados en la ejecución ya habían sido agregados a HDFS.

Cuando sea oportuno, también analizaremos cómo se distribuye la carga de trabajo de una ejecución a lo largo de sus fases Map y Reduce.

Para evitar malgastar los créditos gratis de Google Cloud, todas las pruebas se ejecutaron con un timeout de 10 horas. En algunos casos, esto impidió que podamos establecer exactamente cuánto tiempo llevaría una ejecución, pero igualmente nos dio una cota inferior que a su vez sirvió para acotar el speedup algorítmico. Estos resultados se mostrarán en las tablas con un $>$.

Por último, cabe mencionar que todas las ejecuciones que medimos e incluimos en el informe fueron hechas con el debido cuidado de que el clúster en ese momento no tuviera **ninguna otra carga** que pudiera influenciar los tiempos de cómputo. Lo mismo con las corridas en nuestra PC local.

A. Performance de Regresión Segmentada

Ver tablas III y IV

TABLE III
TIEMPOS (EN SEGUNDOS) DE **REGRESIÓN SEGMENTADA**

Tamaño de entrada	200 MB	1.9 GB	20 GB	57 GB
Serial	2.028	2.202	3.284	$>$ 36.000
2 nodos	603	641	976	2.724
4 nodos	325	353	501	1.348

TABLE IV
SPEEDUP ALGORÍTMICO DE **REGRESIÓN SEGMENTADA**

Tamaño de entrada	200 MB	1.9 GB	20 GB	57 GB
2 nodos	3,4	3,4	3,4	$>$ 13,2
4 nodos	6,2	6,2	6,6	$>$ 26,7

B. Performance de PELT

Ver tablas V y VI

TABLE V
TIEMPOS (EN SEGUNDOS) DE **PELT BASE**

Tamaño de entrada	200 MB	1.9 GB	20 GB	57 GB
Serial	2.240	10.680	$>$ 36.000	$>$ 36.000
2 nodos	654	622	22.560	19.320
4 nodos	369	2.720	23.460	18.540

TABLE VI
SPEEDUP ALGORÍTMICO DE **PELT BASE**

Tamaño de entrada	200 MB	1.9 GB	20 GB	57 GB
2 nodos	3,5	3,1	$>$ 1,6	$>$ 1,9
4 nodos	6,1	3,9	$>$ 1,5	$>$ 1,9

C. Performance de Deal PELT

Ver tablas VII y VIII

D. Análisis de los resultados

Dados los resultados de tiempos y speedup, podemos observar que, en general, la paralelización mejora sustantivamente los tiempos de ejecución, el speedup algorítmico también es significativo en Regresión Segmentada y PELT Deal.

La ejecución de PELT Base sirve como testigo de una implementación que desaprovecha la paralelización, esto se refleja al comparar los resultados con los otros algoritmos utilizados, la diferencia en speedup cuando se utiliza en el problema real (dataset de 57GB) se mantiene por debajo del doble, mientras que en los otros algoritmos se mantiene por encima de 20 veces. Por lo que, lógicamente, la Regresión

TABLE VII
TIEMPOS (EN SEGUNDOS) DE **PELT DEAL**

Tamaño de entrada	**200 MB**	**1.9 GB**	**20 GB**	**57 GB**
Serial	2.334	2.462	5.220	> 36.000
2 nodos	707	753	1.705	2.965
4 nodos	430	430	1.221	1.541

TABLE VIII
SPEEDUP ALGORÍTMICO DE **PELT DEAL**

Tamaño de entrada	**200 MB**	**1.9 GB**	**20 GB**	**57 GB**
2 nodos	3,3	3,3	3,1	$> 12,1$
4 nodos	5,4	5,7	4,3	$> 23,4$

Segmentada y PELT Deal tienen una mejor escalabilidad en comparación con PELT Base.

Discerniendo entre los algoritmos que sí aprovechan la paralelización, cuando se trabajó con el problema real, el speedup algorítmico de Regresión Segmentada con 4 nodos (26.7) es superior al de PELT Deal (23.4) por una diferencia de 3 puntos. De todas formas, esta diferencia no es muy significativa, por lo que es difícil afirmar que la Regresión Segmentada escalaría mejor que PELT Deal, dado que el análisis no es estrictamente concluyente.

Para ampliar la información de performance, planteamos dos propuestas como mejora a futuro:

- Trabajar con un conjunto de datos mayor: Esto podría darnos nueva información sobre el speedup en un problema que necesite más procesamiento. Sin embargo, en la práctica, los datos brindados por Scanntech fueron suficientes para resolver el problema de investigación, como se apreciará en próximas secciones. Avanzar en esta línea implicaría avanzar sobre una problema distinto a la planteado, pero puede dar un mejor conocimiento de la escalabilidad de los algoritmos desarrollados para problemas de mayor porte (Por ejemplo: análisis de mercado de varias categorías a lo largo de un tiempo más prolongado).
- Aumentar la cantidad de nodos: Tanto el diseño de Hadoop, como la infraestructura de Google Cloud permiten aumentar la cantidad de nodos de manera muy sencilla, esto abre la posibilidad de realizar los mismos experimentos con clústeres de mayor porte. En la práctica, esta medida implicaría un gasto que no podemos cubrir, pero sí podría brindar información valiosa del speedup y, en particular, distinguir el punto donde los algoritmos paralelizables comienzan a tener una diferencia sustantiva de performance, si es que la tienen.

E. Tiempos de cada fase MapReduce

Como se adelantó antes, vamos a revisar cómo se distribuyó el tiempo de cómputo en las distintas etapas de los job implementados.

Primero, para hacer comparaciones justas, debe quedar claro las similitudes de los algoritmos en sus maps y reduces. El **primer map** de cada algoritmo hace lo mismo, siempre: JOIN de las tablas involucradas, filtrado de registros y agrupación por clave. El **último map y reduce** es una fase de agregación de resultados ya agrupados por fecha y producto (pocos registros de entrada). En el caso de Regresión Segmentada y PELT base, el primer reduce hace el análisis de tendencias y detección de puntos de cambio, mientras que en Deal PELT, esto se distribuye en ese primer Reduce y la segunda fase Map-Reduce entera. Basándonos en esto, organizamos la tabla IX de forma de que cada fila corresponda a responsabilidades similares o idénticas.

TABLE IX
DISTRIBUCIÓN DEL TIEMPO DE CÓMPUTO EN LAS FASES DE CADA ALGORITMO

Algoritmo	Regresión Segmentada	PELT base	Deal PELT
Primer map	90,54%	30,10%	77,17%
Primer reduce	7,48%	69,13%	18,24%
Segundo map	...	...	1,60%
Segundo reduce	...	...	1,44%
Tercer map	1,84%	0,73%	1,44%
Tercer reduce	0,14%	0,04%	0,11%

VII. RESULTADOS EXPERIMENTALES: EL PROBLEMA

Para esta parte analizaremos los resultados que dieron los distintos algoritmos cuando los corrimos sobre el dataset más grande (el original, de 57 GB).

A. Puntos de quiebre: interpretación de resultados para análisis

El algoritmo fue diseñado para que en su salida final diera, para cada año y categoría de producto, un JSON con las fechas de puntos de quiebre y la cantidad de veces que fue detectada dicha fecha para esos productos. De ese JSON, extrajimos el Top 2 de fechas más comunes (en los pocos casos en los que el conteo del Top 1 fuera muy superior al conteo del Top 2, esta fecha se omite). Esto nos permite hacer un análisis más global de los resultados, abstrayéndonos de productos particulares con comportamiento alejado de la tendencia.

B. ¿Qué esperamos?

Dado que el análisis es sobre la variación de precios a lo largo de los meses de febrero, marzo y abril, esperamos ver cambios de tendencias alrededor del 23 de febrero de 2024, únicamente para las categorías de repelentes e insecticidas. Para la categoría de refrescos, esperamos que los resultados de 2023 sean similares a los de 2024.

C. Resultado de venta total por fecha y categoría

Las figuras 5 y 6 muestran la venta total de cada categoría por día en el año 2023 y en el año 2024 (en pesos uruguayos). Sobre los refrescos podemos observar que las ventas oscilan establemente, con picos en los fines de semana. Esto sucede tanto para el año 2023 y para el 2024, satisfaciéndose con lo esperado para esta categoría.

Para el caso de los insecticidas y repelentes, estos siguen una tendencia muy similar entre ellos en el año 2023, con una distribución uniforme en febrero y marzo, y un decaimiento

Fig. 5. Sumarización de los datos de venta por día y por categoría, para 2023

Fig. 6. Sumarización de los datos de venta por día y por categoría, para 2024

Categoría	Año	Puntos de quiebre detectados	Comentario
Gaseosas	2023	2023-02-28, 2023-03-01, 2023-03-03, 2023-03-08, 2023-03-09, 2023-03-17, 2023-04-10, 2023-04-14, 2023-04-15	Mucha incertidumbre, no detecta nada de forma contundente.
Gaseosas	2024	2024-03-01, 2024-03-06, 2024-03-09	Bastante seguro en esas 3 fechas. 2 son viernes/sábado. ¿Aumento normal de este producto?
Insecticidas	2023	2023-02-04, 2023-02-26, 2023-02-13	También bastante seguro, pero no hay tendencias que expliquen esas fechas.
Insecticidas	2024	2024-02-05, 2024-02-10, 2024-02-16, 2024-02-23, 2024-03-18	Acertado en 23/2, día del pico de venta.
Repelentes	2023	2023-03-03, 2023-03-28, 2023-02-16, 2023-02-14, 2023-02-06, 2023-02-08, 2023-02-12, 2023-03-02, 2023-03-24	Mucha incertidumbre, no detecta nada de forma contundente.
Repelentes	2024	2024-02-25, 2024-03-07	Muy seguro, 25/2 es 2 días después de que los medios empiezan a dar bombo al tema.

Fig. 7. Tabla de fechas obtenidas con Regresión Segmentada, se recuerda que corresponde a cambios en la tendencia de **precios**.

de cara al mes de abril. La diferencia es que la venta de insecticidas es superior en todo momento a la venta de repelentes.

En 2024, la tendencia parecería mantenerse, a excepción de los entornos del 23 de febrero y el 25 de marzo, coincidentes con la invasión y su anuncio en prensa. Este resultado se condice con el planteado en la sección anterior.

Es interesante el aumento en marzo, ya que coincide con una nueva oleada de mosquitos [10].

D. Puntos de cambio obtenidos de usar Regresión Segmentada

Analizando los resultados, sobre la figura 7, vemos que el algoritmo no resultó como esperábamos. Detecta demasiados puntos de cambio cuando los períodos deberían ser más estables. Cuando demuestra ”confianza”, devolviendo pocos puntos, no somos capaces de darles una razón lógica en la mayoría de los casos. ¿Qué pasó con los precios de insecticidas de 2023 para justificar tal seguridad? Lo mismo con las gaseosas en 2024.

Por otro lado, al menos en el momento relativo a la crisis, el algoritmo sí detecta cambios en la tendencia. En 2024, tanto para insecticidas como repelentes, se detectaron cambios cercanos a la fecha donde se disparó la venta. En el caso de repelentes, se detecta el 25/2, 2 días después del boom, por lo que este punto casi seguramente está refiriendo a un aumento de precio producto de una posible escasez. Creemos que es una apuesta segura decir que los precios subieron en esas fechas y que **el algoritmo lo detectó**.

En definitiva, las fechas clave de la invasión son un subconjunto de los puntos encontrados. Es probable que, con un α más severo, se filtren algunos outliers, clarificando el resultado, pero corriendo el riesgo de que se filtren puntos de más.

E. Puntos de cambio obtenidos de usar PELT Base

Repitiendo el análisis, pero sobre la figura 8, encontramos un resultado todavía mejor al anterior. Se aprecia que en ”Repelentes” 2024, para las categorías críticas, el algoritmo detectó con mucha seguridad un cambio de tendencia en dos fechas que ya habíamos mencionado como importantes. No fue así con ”Insecticidas”, dónde únicamente se detectó la fecha de febrero.

Otro aspecto a considerar es que también se mostró ”seguro” en la categoría de gaseosas 2023 y 2024, e insecticidas 2023. No tenemos información para confirmar o desestimar este hallazgo. En cuanto a repelentes 2023, se demuestra muy poca seguridad, además de que detecta días muy cercanos entre sí, lo que no tendría mucho sentido.

Por último, comentamos que en esta ejecución nos sorprendió la cantidad de timeouts dados en el algoritmo. En la configuración se limitó para que ningún algoritmo corra por más de 3 minutos. Correlacionando este dato con la cantidad de datapoints por categoría, concluimos que ese fue el factor predominante en la ocurrencia de dichos timeouts. Veremos en la siguiente sección que PELT Deal logró un comportamiento diferente en este aspecto.

Categoría	Año	Puntos de quiebre detectados	Porcentaje productos con timeout	Comentario
Gaseosas	2023	2023-02-10, 2023-03-11, 2023-04-01	14%	Bastante seguridad, pero no tenemos explicación para esas fechas.
Gaseosas	2024	2024-02-07, 2024-03-14	13%	
Insecticidas	2023	2023-02-04, 2023-03-10, 2023-03-22	4%	
Insecticidas	2024	2024-02-23	2%	Mucha seguridad en la fecha exacta.
Repelentes	2023	2023-02-08, 2023-02-09, 2023-02-12, 2023-02-13, 2023-02-14, 2023-03-13, 2023-03-19	1%	Poca seguridad, da muchos días consecutivos.
Repelentes	2024	2024-02-23, 2024-03-26	1%	Mucha seguridad, en las fechas que ya destacamos.

Fig. 8. Tabla de fechas obtenidas con **PELT base**.

Categoría	Año	Puntos de quiebre detectados	Porcentaje productos con timeout	Comentario
Gaseosas	2023	2023-03-05, 2023-03-10	2%	Bastante confianza, no queda claro por qué.
Gaseosas	2024	2024-02-27, 2024-02-29, 2024-03-07, 2024-03-08, 2024-03-11, 2024-03-19, 2024-03-20, 2024-03-21, 2024-04-07	3%	Poca confianza, sin patrón en los días detectados.
Insecticidas	2023	2023-02-15, 2023-02-28	1%	Bastante confianza, no queda claro por qué.
Insecticidas	2024	2024-02-22, 2024-02-23	2.3%	Muy preciso y con confianza en la fecha clave.
Repelentes	2023	2023-02-08, 2023-02-10, 2023-02-12, 2023-03-10	0.1%	Demasiados puntos cerca del 10/2 y algo de confianza.
Repelentes	2024	2024-03-24, 2024-03-25, 2024-03-27	0.1%	Se pierde el cambio de febrero completamente.

Fig. 9. Tabla de fechas obtenidas con **PELT Deal**.

F. Puntos de cambio obtenidos de usar PELT Paralelizado (Deal)

No detectamos una diferencia sustancial entre los puntos detectados por PELT Deal y PELT base. Ambos detectan cambio en las invasiones de febrero y marzo para ”Repelentes” pero fallan en detectar marzo en ”Insecticidas”. Esto es en sí satisfactorio porque, tal como se mencionó antes, la versión paralelizada de PELT no asegura una convergencia a la segmentación óptima, que PELT normal sí asegura. Se puede concluir que la pérdida en precisión del algoritmo no fue tan grave como para que los resultados finales sean en algún sentido práctico ”peores”.

En el aspecto de timeouts, vemos que se solucionó en gran medida este problema. Al haber repartido los datos en N segmentos, y, por lo tanto, en más Reducers, obtuvimos que el tiempo de la parte de detección de tendencias casi nunca llegó a superar el límite.

VIII. Conclusión y comentarios finales

A lo largo del proyecto hicimos uso de técnicas de HPC que nos ayudaron a resolver el problema planteado. Para ello usamos técnicas de programación paralela basada en descomposición de dominio (en el manejo de claves de MapReduce) mezclada con descomposición funcional (en la implementación alternativa paralela a un algoritmo originalmente secuencial, como fue el caso de PELT). Además, basándonos en las métricas de performance estudiadas, elegimos entre algoritmos que brindaban la misma funcionalidad pero con una implementación radicalmente distinta (estudio de alternativas de JOIN). Con esto logramos llegar a soluciones eficientes y correctas, lo que nos deja un alto grado de satisfacción con el trabajo realizado.

Agradecimientos

Agradecemos a Scanntech UY, en particular al área de desarrollo, por confiar en nosotros con los datos a usar en el proyecto.

Referencias

[1] Akinshin, A. Implementation of an efficient algorithm for changepoint detection: ED-PELT — Andrey Akinshin. https://aakinshin.net/posts/edpelt/.
[2] Amazon Web Services, ¿Cuál es la diferencia entre Hadoop y Spark? aws.amazon.com/es/compare/the-difference-between-hadoop-vs-spark/
[3] Apache Hadoop. https://hadoop.apache.org/.
[4] A Recoba. A Martínez. GitHub - agustin-recoba/hadoop-cluster-docker: Run Hadoop Custer within Docker Containers. GitHub. https://github.com/agustin-recoba/hadoop-cluster-docker.
[5] DataProc - Google Cloud. https://cloud.google.com/dataproc?hl=es.
[6] “Docker Engine overview.” (2024, June 13). Docker Documentation. https://docs.docker.com/engine/.
[7] General-purpose machine family for Compute Engine. Google Cloud. https://cloud.google.com/compute/docs/general-purpose-machines
[8] Invasión de mosquitos: faltan los repelentes por problemas en la importación (Argentina). elobservador.com.uy/argentina/invasion-de-mosquitos-faltan-los-repelentes-por-problemas-en-la-importacion-20241418422.
[9] Killick, R., Fearnhead, P., & Eckley, I. A. (2012). Optimal Detection of Changepoints With a Linear Computational Cost. Journal of the American Statistical Association, 107(500), 1590–1598. https://doi.org/10.1080/01621459.2012.737745.
[10] Magallanes, A. (2024, May 29). Nueva invasión de mosquitos en pleno brote de dengue; casos confirmados suman 158 en todo el país. EL PAIS. elpais.com.uy/informacion/salud/nueva-invasion-de-mosquitos-en-pleno-brote-de-dengue-casos-confirmados-suman-158-en-todo-el-pais.
[11] MSP - Invasión de mosquitos. www.gub.uy/ministerio-salud-publica/comunicacion/comunicados/invasion-mosquitos.
[12] Natarajan, R., & Pednault, E.P. (2002). Segmented Regression Estimators for Massive Data Sets. SDM.
[13] Preocupa la escasez de repelentes mientras crecen los casos de dengue www.ambito.com/uruguay/preocupa-la-escasez-repelentes-mientras-crecen-los-casos-dengue-n5972794.
[14] Subrayado - Una invasión de mosquitos sorprendió este viernes a los habitantes de Montevideo y parte del país. subrayado.com.uy/invasion-mosquitos-especialista-explico-cuanto-durara-el-fenomeno-y-la-posibilidad-que-se-repita-n939329.
[15] Telenoche - Alertan por la invasión de mosquitos en Montevideo telenoche.com.uy/alertan-la-invasion-mosquitos-montevideo-n5363544.
[16] Tickle, S. O., Eckley, I. A., Fearnhead, P., & Haynes, K. (2019). Parallelization of a Common Changepoint Detection Method. Journal of Computational and Graphical Statistics, 29(1), 149–161. https://doi.org/10.1080/10618600.2019.1647216