Centro de Cálculo
INSTITUTO DE COMPUTACIÓN
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA
MONTEVIDEO, URUGUAY

PROYECTO DE GRADO INGENIERÍA EN COMPUTACIÓN

PROCESAMIENTO DE GRANDES VOLÚMENES DE DATOS DE MOVILIDAD URBANA

Octubre 2017

Autor
Jonathan Martín Denis García

Supervisores
Sergio Nesmachnow
Renzo Massobrio

Agradecimientos

Deseo agradecer especialmente mi familia: mi esposa Inés, mis padres Silvia y Anibal, mis hermanas Paula y Belén, que son las personas que me dieron su apoyo incondicional en todo momento, en especial en los momentos más difíciles.

También vaya el agradecimiento a mis amigos, quienes siempre supieron comprender la importancia de esta instancia para mí, respetando y dando ánimo.

A todos mis compañeros de trabajo y a mis jefas, quienes me dieron su total apoyo para poder continuar con mis estudios contribuyendo a compatibilizar la actividad estudiantil con el desarrollo laboral.

Resumen

El presente trabajo se enmarca en el uso de sistemas de transporte inteligente, analizando un caso particular: el sitema de transporte de Montevideo. Se busca realizar un análisis y procesamiento de datos producidos por el sistema para generar herramientas útiles para la toma de decisiones, con el fin de mejorar la experiencia de los usuarios.

Se toman los registros de los dispositivos de GPS presentes en los ómnibus de la ciudad de Montevideo, para obtener datos de velocidades promedio según cada línea. Al tratarse de un gran volumen de datos, se utilizan algoritmos paralelos para la obtención resultados en tiempos aceptables.

Finalmente se procede a la creación de una herramienta web para la visualización de los resultados, valiéndose de la información geográfica para trabajar sobre mapas, utilizando herramientas de código abierto como lo son postGIS, GeoServer y OpenLayers.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

El contexto actual de desarrollos tecnológicos, especialmente en cuanto a la infraestructura disponible para la generación y el procesamiento de datos, demuestra grandes avances en un corto período de tiempo. Por un lado existen dispositivos con gran capacidad de cómputo, a través de los cuales se pueden analizar grandes volúmenes de datos. Por otro lado se ha masificado el acceso a diferentes dispositivos generadores de datos, destacándose los dispositivos móviles, entre los que se incluyen los dispositivos para navegación geográfica mediante el uso de GPS.

Algunos de los avances tecnológicos mencionados en el párrafo anterior han sido utilizados en diferentes ciudades alrededor del mundo con el fin de generar datos para el análisis de sus sistemas de transporte, abordando temáticas que van desde estudios en la movilidad de sus usuarios hasta diferentes soluciones para la optimización de los recursos. Usualmente este tipo de sistemas de transporte son conocidos como ITS, Intelligent Transportation Systems, por sus siglas en inglés (sistemas inteligentes de transporte).

El sistema de transporte metropolitano (STM) es el caso de estudio que se aborda en este proyecto. Se trata de un ITS que incorpora el uso de nuevas tecnologías en las diferentes unidades de transporte colectivo de la ciudad de Montevideo, con el fin de lograr mejoras en el servicio que se brinda a la población de la ciudad.

La recolección de datos aportados por los dispositivos instalados en las unidades de transporte, junto con el análisis e interpretación de los mismos, resulta ser un mecanismo que sirve para comprender de mejor forma cómo funciona el STM, dónde hay falencias y descubrir oportunidades de mejoras que puedan ser aplicadas sobre dicho sistema.

Como caso de estudio de este proyecto se procesan y analizan los datos obtenidos de las unidades de transporte pertenecientes al STM, recolectados durante el año 2015. El conjunto de datos utilizados en el proyecto incluye

mediciones de posiciones de las unidades de transporte mediante el uso de dispositivos GPS y el tiempo en el que fueron tomadas, por lo que se pueden calcular distancias, tiempo de los recorridos y sus velocidades.

Para la etapa de procesamiento de los datos se utilizan técnicas de programación paralela, buscando reducir los tiempos de procesamiento de los datos, con respecto a soluciones no paralelas. La estrategia seleccionada para el proyecto consiste en el uso de colecciones paralelas del lenguaje Scala, que permite el uso de programación multihilada de forma rápida.

El proyecto también cuenta con otra contribución especialmente importante, referida a la visualización de resultados: el diseño e implementación de una herramienta que permite mostrar la información generada en la etapa de procesamiento. Para la visualización de resultados, se hace uso de una característica importante de los datos que se manejan, que es la georreferenciación.

Para la implementación de esta herramienta se utilizan diversas bibliotecas y servicios, entre los que se destacan PostGIS, como base de datos geográficos, Geoserver, como servidor web con webservices geográficos, y Openlayers para la creación de mapas en sitios web.

En primera instancia se realizó una investigación acerca de los datos disponibles, tanto de los que fueron aportados por la intendencia de Montevideo, como aquellos datos que forman parte del catálogo de datos abiertos del gobierno uruguayo. Luego, se investigaron diferentes herramientas para el procesamiento de los datos y para la visualización de la información resultante de la etapa de procesamiento. Finalmente, se procedió al análisis, diseño e implementación de una solución para el problema abordado.

El resto del informe se organiza de la siguiente manera. En el capítulo 2 se introduce al paradigma de la programación paralela, especialmente en la programación multihilada, y se explica el funcionamiento del lenguaje Scala, el cual es utilizado en el proyecto. En el capítulo 3 se exponen las herramientas utilizadas para la generación de los datos geográficos y el desarrollo de la herramienta de visualización de los resultados. Luego, en el capítulo 4 se plantea el problema a resolver, mostrando el marco en el cual se desarrolla este trabajo, para proceder a mostrar diferentes proyectos relacionados a los ITS. En el capítulo 5 se presentan las diferentes arquitecturas y tecnologías usadas en cada etapa. Más adelante, en el capítulo 6, se presentan los detalles de las diferentes implementaciones usadas en el proyecto. En el capítulo 7 se detallan los análisis experimentales realizados, así como los resultados obtenidos. Finalmente, en el capítulo 8, se abordan las conclusiones del trabajo y se plantean posibles acciones a realizar en el futuro.

Capítulo 2

Programación paralela

2.1. Introducción

La programación paralela es el proceso de desarrollar e implementar programas paralelos [2], lo cual consiste en la utilización de programación concurrente de forma de ejecutar sentencias en forma simultánea, para poder abordar problemas de alta complejidad utilizando varias unidades de cómputo al mismo tiempo. De esta forma es posible abordar problemas de mayor escala, tanto en complejidad de cálculos como en cantidad de datos que manejan.

La estrategia al utilizar programación paralela consiste en dividir el problema inicial en sub-problemas de menor tamaño, que puedan ser resueltos en forma paralela, manejando las comunicaciones con un esquema de programación concurrente para mantener la consistencia de los datos.

Los diferentes procesos que trabajan con cada una de las partes del problema a atacar deben utilizar mecanismos de sincronización, los cuales consisten en el uso de recursos compartidos, como puede ser memoria compartida, o el pasaje de mensajes utilizando memoria distribuida. Los sistemas de computación que dan soporte a este tipo de programación se denominan computadores paralelos. A su vez, los programas que ejecutan en un computador paralelo utilizando múltiples procesos simultáneos se denominan programas paralelos, diferenciándose de los programas secuenciales tradicionales.

2.2. Computación de alta performance y de alto rendimiento

Existen fundamentalmente dos modelos de computación paralela/distribuida, donde cada modelo pone el foco en un objetivo diferente. Por un lado se en-

cuentran los sistemas de computación de alta performance (HPC, por sus siglas en inglés High Performance Computing) y por otro los sistemas de computación de alto rendimiento (HTC, por sus siglas en inglés High Throughput Computing) [3].

2.2.1. HPC

Los sistemas HPC hacen especial énfasis en la eficiencia, considerando el número de operaciones realizadas por unidad de tiempo [4]. Este tipo de sistemas son utilizados usualmente en computación científica dado que permiten la realización de cálculos más complejos en menos tiempo. Gracias a los avances tecnológicos la velocidad de procesamiento de este tipo de sistemas ha ido en aumento así como su disponibilidad.

El desarrollo de la tecnología grid y cloud permite compartir los recursos informáticos a nivel global para lograr un trabajo colaborativo en forma conjunta, permitiendo crear sistemas HPC distribuidos con alto poder de cómputo. Brinda la posibilidad de gestionar y distribuir el poder de cálculo a una escala mayor que los entornos locales. Una de las principales ventajas es que se logra una alta capacidad de cómputo, especialmente útil en ámbitos académicos e industriales.

Un ejemplo de sistema HPC en nuestro país es el Cluster FING [5]. Es parte de la infraestructura de cómputo de alto desempeño de la Facultad de Ingeniería, buscando dar una opción computacional eficiente para la resolución de problemas complejos. Este cluster tiene el mayor poder de cómputo disponible en el país.

2.3. Computación paralela

Tanto los sistemas HPC como los sistemas HTC se enfocan en la eficiencia desde diferentes ángulos. HPC evalúa la eficiencia en términos de tiempos de ejecución y uso de recursos, mientras que HTC evalúa la eficiencia a partir de la cantidad de tareas y usuarios que se logran atender. Para dar soporte a la creciente demanda de poder de procesamiento también se debe tener en consideración aspectos de confiabilidad, buscando altos niveles de calidad de los servicios incluso ante escenarios donde puedan ocurrir fallas. Este último objetivo se puede lograr mediante mecanismos de tolerancia a fallos [6].

Para poder cumplir con los objetivos que tienen los sistemas HPC y HTC, se requiere adaptar los modelos de programación para poder utilizar diferentes recursos físicos y diferentes modelos de trabajo (paralelo, distribuido,

etc.). También es importante proveer cierta flexibilidad, permitiendo que las aplicaciones programadas trabajen en ambientes de HPC y de HTC.

En las condiciones antes mencionadas, fue desarrollado el procesamiento paralelo, principalmente en base a estudios de universidades.

Existen diferentes grados de paralelismo [7], desde paralelismo a nivel de bits o de instrucciones (pipelining por ejemplo), hasta el paralelismo distribuido o cloud computing, que se ha desarrollado gracias al avance de Internet y de las redes de computadoras. Existen otros niveles intermedios de paralelismo que se puede dar a nivel de hilos de ejecución (multithreading) o sobre supercomputadores (a nivel de procesos de ejecución).

La principal ventaja del procesamiento paralelo es la mayor capacidad de cómputo que otorga. Con ello se logra un aumento en la complejidad de los problemas que se pueden abordar, ampliando los objetivos y alcances. Otra gran ventaja del procesamiento paralelo es que logra reducir los tiempos de procesamiento, lo cual permite reducir costos y dar más tiempo a otras etapas de desarrollo o ejecución. También es importante observar que el procesamiento paralelo puede permitir un mejor aprovechamiento de los recursos disponibles, disminuyendo los tiempos en que dichos recursos no se utilizan.

2.3.1. Objetivos de la computación paralela

Mediante el uso de la computación paralela se busca la resolución de problemas complejos y de gran dimensión, en forma eficiente aprovechando al máximo los recursos de cómputo disponibles [7].

Para que el paradigma de computación paralela resulte atractivo debe contar con mecanismos que hagan transparente al usuario los mecanismos de comunicación y sincronización. También es necesario que sea robusto y confiable, a través del manejo de excepciones y tolerancia a fallos. Otras características deseables a nivel de usuario son la portabilidad, permitiendo ejecuciones en entornos heterogéneos dando soporte a lenguajes de programación de alto nivel.

Se pueden distinguir dos enfoques de paralelismo computacional: paralelismo implícito, en la cual el usuario no controla el procesamiento, y paralelismo explícito, en el cual el programador es responsable del manejo paralelo de los recursos. Este último paradigma permite mayor manejo del paralelismo dependiendo del problema a afrontar, a través de la lógica del programa que se implementa.

En base a los dos enfoques antes mencionados, surgen diferentes estrategias de programación paralela. En el caso del paralelismo implícito, se parte de un código pre-existente y se le aplican algunos cambios menores, dejando la

optimización a cargo de los compiladores. Cuando se trabaja con paralelismo explícito, se puede partir del uso de bibliotecas ya paralelizadas convirtiendo un código ya existente, o se puede hacer un rediseño de código implementando una nueva lógica utilizando bibliotecas de programación paralela.

2.3.2. Niveles de paralelismo

La aplicación de estrategias de computación paralelas puede ser en diferentes niveles según el problema que se desea resolver.

El nivel más bajo de paralelismo se puede ver a nivel de intrainstrucción, mediante el uso de hardware, dividiendo cada instrucción en sub-partes que son ejecutadas por diferentes componentes del hardware. Un segundo nivel es el paralelismo de interinstrucción, en dónde el paralelismo lo resuelve el sistema operativo, administrando los recursos en forma lo más óptima posible. El tercer nivel es de procedimientos o tareas, dónde el programador es el encargado implementar las estrategias paralelas de los algoritmos. Finalmente el cuarto nivel de paralelismo es a nivel de programas o trabajos. Este cuarto nivel consiste en orquestar la ejecución diferentes programas en forma paralela, manejando la comunicación y sincronización entre los programas o trabajos.

2.3.3. Diseño de algoritmos paralelos

El diseño de algoritmos paralelos consta de cuatro etapas: descomposición, asignación, orquestación y mapeo.

La etapa de descomposición consiste en identificar las partes del trabajo que pueden realizarse en forma paralela de aquellas que se deben realizar en forma secuencial. En base a esa identificación, se decide el nivel de paralelismo y la forma en que se usará la concurrencia. En esta etapa se toman decisiones sobre la cantidad de tareas y los criterios de división y utilización de recursos.

En la etapa de asignación se divide el trabajo y los datos en los diferentes procesadores. Se deben definir los criterios de asignación, que pueden ser estáticos o dinámicos y se aplican mecanismos para el balance de cargas entre los procesadores con el objetivo de reducir los costos de comunicación y sincronización.

La tercera etapa es la orquestación. En esta etapa se toman las decisiones sobre cómo realizar los accesos a los datos, y las comunicaciones y sincronizaciones entre los procesos. Se debe decidir el tipo de arquitectura a utilizar, el modelo de programación y el lenguaje o bibliotecas de programación que se debe usar. Las estructuras de datos se deben resolver en esta etapa.

Finalmente en la etapa de mapeo o scheduling se asignan los recursos de cómputo a los procesos que se ejecutan en forma paralela. Para ello se deben definir los criterios de asignación, que pueden ser por desempeño, por grado de utilización de cada proceso, o por criterios para economizar los costos de comunicación y sincronización. Estos criterios pueden ser estáticos o dinámicos.

El lenguaje de programación elegido debe dar mecanismos para definir, controlar y sincronizar las tareas, permitiendo especificar el control del flujo de ejecución de las tareas y el particionamiento de los datos. Para cumplir estos requisitos existen varias herramientas, como puede ser el uso de fork y join en el lenguaje C, que permite crear diferentes tareas manteniendo referencias a la tarea padre. En cuanto a la sincronización, existen mecanismos como los semáforos que permiten coordinar el uso de recursos compartidos. El particionamiento de los datos es realizado normalmente por el diseñador del algoritmo paralelo.

Es importante destacar que en algunas ocasiones al crear algoritmos paralelos se puede partir de un algoritmo secuencial para ser "paralelizado", pero ésta no siempre es una buena decisión. Existen situaciones en las cuales es necesario el diseño de un nuevo algoritmo pensado especialmente para ser paralelo, el cual puede resultar muy diferente al algoritmo secuencial inicial.

2.4. Técnicas de programación paralela

Las diferentes técnicas de programación paralela buscan aplicar estrategias de descomposición o particionamiento de los datos y de las instrucciones de cómputo, buscando dividir un problema en sub-problemas más pequeños de menor complejidad y que puedan ser ejecutados en forma paralela. Usualmente se busca una división equitativa de los datos y/o de los cálculos a ejecutar.

Dependiendo del enfoque que se realice, priorizando la división de datos o de tareas a realizar, se obtienen diferentes técnicas de programación paralela. Las técnicas más utilizadas en la actualidad son las de descomposición de dominio y descomposición funcional que se describen a continuación.

2.4.1. Descomposición de dominio

La técnica de descomposición de dominio se basa en el particionamiento de los datos del problema. Busca dividir los datos en partes más pequeñas, de forma que las unidades resultantes tengan tamaños similares. Luego divide

los cálculos que se deben realizar, asociando las diferentes operaciones con los datos sobre los cuales deben operar.

Los datos a dividir pueden ser los de entrada del programa, los de la salida que genera el programa, o se pueden realizar divisiones de datos intermedios calculados por el programa en ejecución.

No existe una regla general para realizar la división de los datos, pero usualmente se toman como punto de partida algunas divisiones que puedan resultar obvias según la estructura del problema, o de los propios datos a analizar. Otro de los puntos en los que se hace énfasis al momento de dividir los datos es en la cantidad de accesos a cierta estructura o su tamaño, buscando dividir los conjuntos de datos más grandes o los que son accedidos más frecuentemente.

2.4.2. Descomposición funcional

La técnica de descomposición funcional se concentra en particionar las operaciones del problema. Busca la división del procesamiento en tareas disjuntas que puedan ser ejecutadas simultáneamente.

Después de realizar la división de tareas, se deben analizar los datos que utiliza cada tarea definida anteriormente, donde surgen dos tipos de particionamiento. El particionamiento completo se da cuando los datos que utilizan las diferentes tareas son disjuntos. En caso contrario se tiene un particionamiento incompleto, el cual requiere replicar los datos o generar mecanismos de comunicación entre los procesos para sincronizar el uso de los datos.

2.5. Evaluación de desempeño de algoritmos paralelos

La evaluación de desempeño de algoritmos paralelos permite evaluar y comparar los algoritmos paralelos con los algoritmos seriales. En forma intuitiva surgen dos factores principales para evaluar la performance: el tiempo de ejecución y el uso de los recursos disponibles.

2.5.1. Tiempo de ejecución

El tiempo de ejecución es la medida de rendimiento de un programa para resolver un problema, por lo cual es un útil para evaluar algoritmos paralelos. Es por eso que habitualmente se usa el tiempo total de ejecución como medida de desempeño. Se debe tener en cuenta que el almacenamiento y trasmisión

de datos entre los procesos puede influir en forma negativa en el tiempo de ejecución de un algoritmo paralelo. En el caso de los programas paralelos, el tiempo de ejecución se mide desde el comienzo de la ejecución del primer proceso hasta el momento en que finaliza la ejecución del último proceso.

Se debe considerar que el tiempo de ejecución se divide en tres componentes: tiempo de procesamiento efectivo, tiempo de comunicación y tiempo ocioso. El primero depende de la complejidad y dimensión del problema, de la cantidad de tareas y de los recursos de cómputo disponibles. El tiempo de comunicación depende la cantidad de unidades de cómputo, y de los canales que utilicen esas unidades de cómputo para la comunicación entre procesos. Está compuesto por el tiempo que insume establecer la comunicación, tiempo de latencia, y por el tiempo que demanda la transferencia de la información. El tiempo ocioso de ejecución es el que se debe procurar minimizar para aumentar el rendimiento de un programa. Surge como una consecuencia de la ausencia de recursos de cómputo disponibles o por ausencia de datos sobre los que operar. Una forma de mitigar el impacto que genera el tiempo ocioso consiste en usar estrategias de balanceo de carga, de forma de distribuir los datos o los cálculos de forma apropiada.

2.5.2. Mejora de desempeño: speedup

Los algoritmos paralelos buscan un mejor aprovechamiento de los recursos de cómputo disponibles, por lo que medir el uso de dichos recursos y la capacidad de su uso en la resolución de los problemas, resulta en una buena métrica de evaluación de algoritmos paralelos.

El speedup es una medida de la mejora en la performance de una aplicación al aumentar la cantidad de recursos de cómputo disponibles, en comparación al rendimiento que se tiene al utilizar un único procesador [8].

Una de las variantes de speed up es el Speedup absoluto, el cual se calcula como el cociente del tiempo de ejecución del mejor algoritmo secuencial que resuelve el problema (T_0), sobre el tiempo total de ejecución del algoritmo paralelo al ser ejecutado sobre N recursos de cómputo (T_N): $S_N = T_0/T_N$. Otra variante de speedup es el Speedup algorítmico, que se calcula como el cociente del tiempo de ejecución del algoritmo sobre un único recurso de cómputo, simulando un comportamiento secuencial (T_1), sobre el tiempo total de ejecución del algoritmo paralelo al ser ejecutado sobre N recursos de cómputo (T_N): $S_N = T_1/T_N$.

El speedup algorítmico es el más usado en la práctica para evaluar la mejora de desempeño de algoritmos paralelos porque es difícil conocer el mejor algoritmo serial que resuelva el problema para poder estimar el speedup absoluto.

Al momento de la evaluación se debe tener en consideración la configuración del computador paralelo que se utiliza, teniendo en cuenta el hardware disponible. En caso de trabajar con recursos heterogéneos, es conveniente dar distintos grados de importancia a los diferentes equipos para obtener una comparación más justa entre los algoritmos.

Al momento de evaluar un algoritmo paralelo, se busca que su speedup sea lineal. Esto quiere decir que al aumentar la cantidad de unidades de cómputo, se obtenga una mejora lineal en la performance del algoritmo. Por ejemplo, al aumentar al doble las unidades de cómputo, que la performance del algoritmos también aumente al doble.

La realidad muestra que es difícil llegar al escenario de un speedup lineal, y que habitualmente se alcanzan resultados con un speedup sub-lineal. Esto se puede dar por varios factores, entre los cuales se encuentran las demoras en las comunicaciones entre procesos, la existencia de partes del procesamiento que no son paralelizables, etc. Incluso se puede llegar a escenarios en los cuales agregar procesadores resulte contraproducente para la performance.

2.6. Programación multithreading

La programación multithreading se basa en el concepto de hilos (threads en inglés), los cuales son una unidad básica de uso de CPU [8]. Todo hilo pertenece a un proceso, pero cada proceso puede estar compuesto por más de un hilo. Todos los hilos que pertenecen a un mismo proceso comparten la memoria asignada dicho proceso así como los archivos y el código ejecutable. Sin embargo, cada hilo cuenta con su propio conjunto de registros y stack, y además cada hilo tiene un identificador diferente, llamado TID (Thread IDentifier).

En la figura 2.1 se presenta una comparación entre un proceso compuesto por un único hilo y otro multithreading.

Algunas de las principales ventajas del uso de hilos radican en que son más simples que un proceso, por lo que resultan más livianos para el sistema operativo. Además, al tener varios elementos en común, los cambios de contexto son más simples ya que para cambiar de un hilo a otro se pueden conservar las secciones en común. Otra ventaja es el hecho de contar con memoria compartida, la cual simplifica la sincronización de los diferentes hilos pertenecientes a un mismo proceso.

La principal desventaja que tiene el uso de hilos de ejecución es la tolerancia a fallos, ya que si un hilo cae hace caer al proceso y por ende a los demás hilos de ejecución. También se puede ver como una desventaja el hecho de compartir tantos elementos, ya que implica una mayor cautela a la hora de

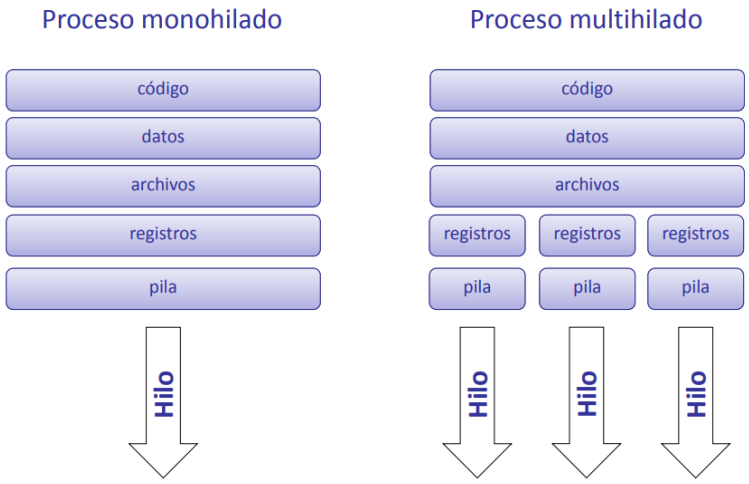


Figura 2.1: proceso de un hilo vs multithread.

programar.

Los hilos pueden ser implementados a nivel de usuario, en bibliotecas que den soporte a la creación, administración y planificación de hilos de ejecución sin soporte del sistema operativo, o pueden ser implementados a nivel del núcleo del sistema operativo, que se encarga de dar soporte a la creación, planificación y administración de threads.

En el caso de manejar programación concurrente multithreading, se debe tener en consideración la existencia de secciones de datos que son compartidas, generando mecanismos de sincronización para garantizar la integridad de dichos datos. Usualmente los diferentes sistemas operativos cuentan con herramientas para resolver este tipo de problemas, a través del manejo de bloqueos.

El sistema operativo intenta balancear la carga migrando hilos entre los diferentes núcleos del procesador, lo cual tiene un costo asociado al reinicio del pipeline de ejecución del hilo, por lo que se intenta evitar. Para reducir los cambios de hilos de ejecución entre los núcleos del procesador, el programador puede definir, mediante algunas bibliotecas, la afinidad de cada hilo a ciertos núcleos que serán los únicos aptos para ejecutarlo.

2.7. Programación paralela en Scala

Scala es un lenguaje de programación que propone una integración entre la programación orientada a objetos y la programación funcional, procurando potenciar las virtudes de cada uno de estos paradigmas [9]. La sintaxis

de Scala es similar a la que utiliza Java, con algunas simplificaciones y nuevos componentes. Los programas desarrollados en Scala se ejecutan sobre la máquina virtual Java (en inglés Java Virtual Machine, JVM) y son compatibles con las aplicaciones y librerías implementadas con el lenguaje Java.

El lenguaje de programación Scala presenta una modalidad para dar soporte a la programación paralela a través de las colecciones paralelas (Parallel Collections) [10].

2.7.1. Parallel Collections

Las Parallel Collections son la implementación que brinda el lenguaje Scala para simplificar el desarrollo de aplicaciones paralelas, generando una abstracción de los detalles de bajo nivel para que los usuarios puedan trabajar a alto nivel en forma simple e intuitiva [10]. Lo que plantea Scala es el uso de colecciones, que usualmente permiten el manejo y procesamiento de sus elementos en forma paralela e independiente, para optimizar las ejecuciones. De esta forma se busca acercar la programación paralela a usuarios que normalmente no están acostumbrados a este paradigma.

La librería de Parallel Collections de Scala está inspirada e integrada con la librería de colecciones secuenciales, de forma que se provee una contraparte paralelizada a un número importante de estructuras de datos de la librería de colecciones secuenciales de Scala.

Si bien las Parallel Collections se parecen mucho a las colecciones secuenciales, es importante indicar que se rigen por los efectos habituales de la programación paralela, de forma que pueden haber efectos secundarios no deseados o resultados no deterministas.

La forma de trabajo paralelo de la librería Parallel Collections consiste en dividir en forma recursiva una colección dada, aplicando la operación solicitada en forma paralela sobre cada partición de la colección, recombinando los resultados al finalizar el procesamiento. Estas ejecuciones se dan en forma concurrente y fuera de orden, lo cual genera problemas cuando se tienen operaciones con efectos secundarios u operaciones no asociativas.

Un clásico ejemplo de operación con efectos secundarios es la escritura o modificación de variables globales. En este caso cada hilo de ejecución va modificando la variable y pisando el resultado que haya escrito algún otro hilo, obteniéndose un resultado no determinista. Con las operaciones no asociativas sucede algo similar, ya que no se puede saber el orden en que se procesarán los elementos de la colección. Un ejemplo de problema con operación no asociativa es la aplicación de una reducción, realizando la resta elemento a elemento de la lista. En este caso, es importante el orden en que se restan los elementos, por lo que para dos ejecuciones diferentes se puede

llegar a resultados diferentes.

2.7.2. Array Paralelo

El Array Paralelo (ParArray) de Scala es el tipo de colección utilizado en el proyecto. Se trata de una secuencia de elementos contiguos lineales [11]. Esto permite el acceso y actualización de los elementos en forma eficiente al modificar la estructura subyacente (un array). La iteración sobre sus elementos es también eficiente por esta misma razón. La principal característica de los Arrays Paralelos es que tienen un tamaño constante, al igual que los Arrays secuenciales.

Para realizar el procesamiento de un Arary Paralelo, Scala utiliza internamente Splitters que dividen el Array original y crea nuevos Splitters con sus índices actualizados. Al culminar el procesamiento de los elementos se utilizan Combiners para agrupar los resultados. Los Combiners tienen una tarea más pesada, ya que en la mayoría de los métodos transformadores no se sabe la cantidad de elementos de salida, por lo que cada Combiner funciona como un array buffer. Los diferentes procesos añaden elementos a los combiners de arrays que les corresponden en forma separada, para luego ser combinados al encadenar los arrays internos. Cada array correspondiente a cada Combiner se crea en memoria para ir completando sus elementos una vez que se conoce el número total de elementos necesarios. Es por este funcionamiento que los métodos transformadores resultan más costosos que los métodos de acceso.

2.7.3. Creación de Parallel Collections

Las Parallel Collections pueden ser creadas mediante dos mecanimsos diferentes [10]. El primer mecanismo a través de la invoación de la función *new* aplicado al tipo de colección paralela deseada (ver figura 2.2), y el segundo mecanismo es a partir de la paralelización de una colección secuencial previamente existente utilizando la función *par* (ver figura 2.3).

```
import scala.collection.parallel.immutable.ParVector
val pv = new ParVector[Int]
```

Figura 2.2: Creación Parallel Collection.

La característica de conversión entre colecciones secuenciales y paralelas es bidireccional. Así como existe la función *par* para crear una colección paralela a partir de una colección secuencial, Scala también cuenta con la

```
val pv = Vector(1,2,3,4,5,6,7,8,9).par
```

Figura 2.3: Creación Parallel Collection a partir de colección secuencial.

función *sec* que permite convertir una colección paralela en una colección secuencial.

En el caso de las colecciones que son inherentemente secuenciales, como las listas, colas y streams, no se cuenta con un tipo paralelo exclusivo. El mecanismo que presenta Scala para trabajar en forma paralela con estos tipos de colecciones consiste en realizar una copia de todos sus elementos a una colección paralela de otro tipo como puede ser un vector paralelo. La desventaja de este mecanismo consiste en un aumento en el costo del procesamiento debido al aumento en la carga de trabajo que conlleva la copia de los elementos.

En el caso de los Arrays Paralelos al invocar el método *sec* son convertidos al tipo de colección ArraySeq, que es la contraparte secuencial del ParArray. A bajo nivel, el ArraySeq utiliza el mismo array que había sido creado por el array paralelo, siendo una opción altamente eficiente.

2.7.4. Configuración de colecciones paralelas

Entre las simplificaciones que implica el uso de colecciones paralelas, se tiene la planificación y distribución de carga entre los procesadores, de lo que se encarga el objeto "Task support"[12]. Cada colección paralela cuenta con uno de estos objetos, el cual mantiene una referencia a los hilos de ejecución y se encarga de resolver la división de las tareas y la asignación de dichas tareas a cada uno los hilos de ejecución que administra. Estos hilos son mantenidos en un pool, para evitar la creación y destrucción de hilos nuevos, aprovechando los ya existentes y que se encuentran dormidos, optimizando así su utilización.

Cada colección cuenta por defecto con un "Task support"que depende del contexto de ejecución("ExecutionContextTaskSupport") el cual está dado por la librería "Scala.concurrent", usando el pool de hilos (thread pool) de esa librería. También se cuenta con otros dos tipo de "Task support": "ForkJoinTaskSupport", y "ThreadPoolTaskSupport". El primero utiliza un pool basado en las operaciones fork y join y puede ser utilizado en versiones de la JVM 1.6 o superiores. El segundo tipo es menos eficiente y se usa solamente para versiones inferiores de la JVM en donde no se puede hacer uso del primero.

Cuando se define una colección paralela, el usuario puede optar por asignarle un "Task support"nuevo o dejar el que tiene por defecto. En el caso de seleccionar uno nuevo, además se puede seleccionar el nivel de paralelismo que se desea utilizar. Esto implica indicar la cantidad de hilos o threads que se crearán y se colocarán en el pool de ejecución.

Capítulo 3

Herramientas para uso de datos geográficos

Cada vez es más común encontrarse en la industria del software con requerimientos que implican el manejo de datos geográficos. Este tipo de datos tiene varias características que lo hacen diferente a los tipos más comunes soportados por cualquier lenguaje de programación como pueden ser los número enteros, los booleanos o las cadenas de texto. Esto implica una serie de desafíos que se deben abordar en las diferentes etapas del desarrollo de una aplicación geográfica.

En esta sección se abordan conceptos fundamentales de geografía y de sistemas geográficos, para luego presentar algunas de las herramientas más utilizadas en la industria del software.

3.1. Introducción

Para empezar, el desarrollador debe familiarizarse con los conceptos geográficos, como qué es un mapa, que es la cartografía, y algunos más que veremos a continuación, basados en los conceptos expuestos en el curso "Taller de Sistemas de Información Geográficos Empresariales"[1] .

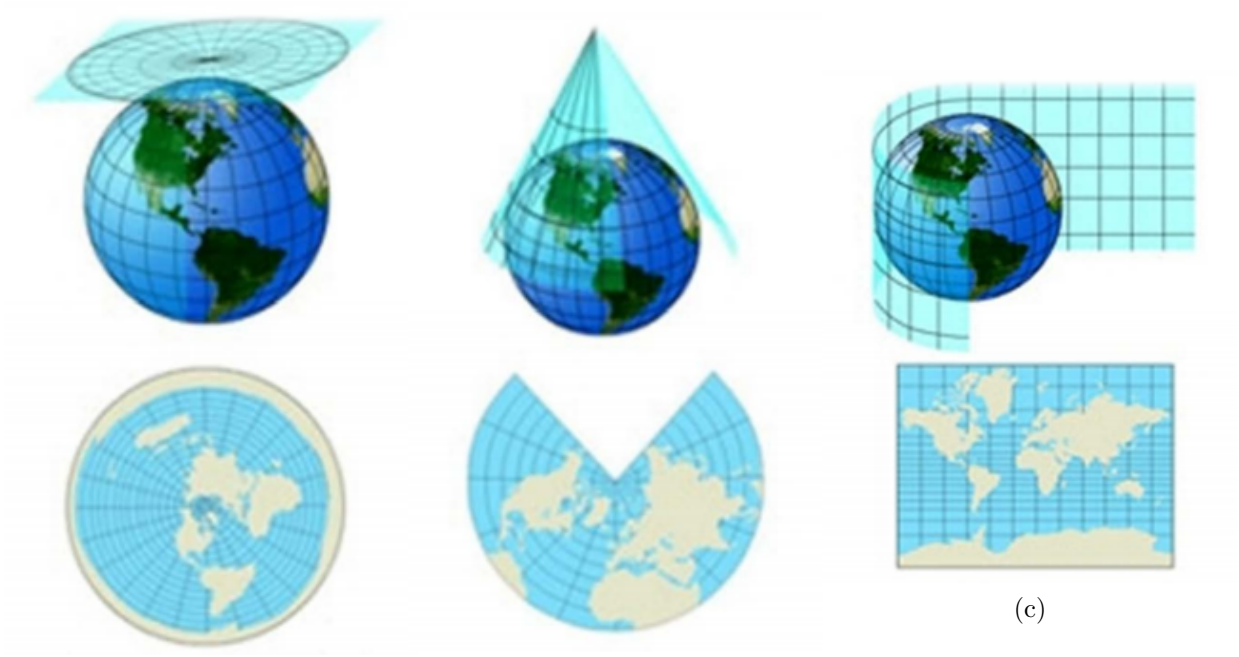
Mapa Del diccionario de la lengua española obtenemos la siguiente definición: "Representación geográfica de una parte de la superficie terrestre, en la que se da información relativa a una ciencia determinada". Lo importante es observar que un mapa es la forma que se tiene para representar la información geográfica en un plano y que los datos que se muestran en el mismo pueden variar dependiendo de la finalidad que éste tenga.

Conceptos de cartografía El formato de la Tierra responde a un geoide, el cual es una figura irregular. Sin embrago para modelar la tierra se utilizan algunas aproximaciones mediante figuras regulares como pueden ser una esfera o un elipsoide. Existen varias aproximaciones diferentes, pero la más usada en la actualidad es la que se realiza a través del WGS84 (Sistema Geodésico Mundial 1984 por su sigla en inglés). El sistema antes mencionado es un estándar en geodesia, cartografía, y navegación, que data de 1984. Este sistema consiste en un patrón matemático de tres dimensiones que representa la tierra por medio de un elipsoide. Otro concepto muy importante es el de Datum. Cada Datum define una serie de parámetros que sirven para relacionar un elipsoide al geoide que representan.

Coordenadas La georreferenciación consiste en asignarle a un dato, ciertas coordenadas en la tierra. Para ello es necesario utilizar un sistema de coordenadas que pueden identificar un punto de la tierra en forma exacta. Existen diferentes tipo de coordenadas, tales como coordenadas geográficas o coordenadas planas cartesianas.

Proyección geográfica Una proyección geográfica es una relación biunívoca entre los puntos de la superficie terrestre y los puntos de una superficie plana, como puede ser un mapa. Existen principalmente tres tipos de proyecciones: planas, cónicas o cilíndricas. En la primera (Figura 3.1a), se coloca un plano tangente por un punto de contacto a la Tierra y se proyectan las coordenadas en el plano. En la segunda (Figura 3.1b) se utiliza un cono tangente a la Tierra mediante una circunferencia y se proyectan en él las coordenadas. Por último, en la proyección cilíndrica (Figura 3.1c) se hace pasar un cilindro en forma tangente a una circunferencia de la Tierra o secante a dos circunferencias y se proyectan las coordenadas terrestres en él.

Universal Transverse Mercator (UTM) Se trata de un sistema de referencia geográfico basado en coordenadas cartesianas que usa la proyección transversa de Mercator. Tiene la característica de que sus coordenadas se miden en metros, avanzando las coordenadas x hacia el Este y las coordenadas y hacia el Norte. Este sistema divide 60 zonas o husos horarios (cada uno con 6 grados de longitud), que van desde la zona 80S hasta la 84N. También cuenta con 20 bandas de 8 grados de latitud, que se denominan a partir de la letra C hasta la letra X (sin contar con las letras I, N ni Ñ). Como se puede observar en la figura 3.2 Montevideo se encuentra en el cuadrante 21H.



(a) (b) (c)

Figura 3.1: Proyecciones geográficas [1]

Modelos de datos geográficos Existen varias formas de modelar los datos geográficos. Uno de los más usados es el ráster, el cual consiste en una matriz bidimensional en donde cada pixel posee un valor numérico con el cual se representa un color. Es utilizado para imágenes, como puede ser una foto aérea por ejemplo. Otro modelo es el vectorial, el cual consiste en representar en forma geométrica las características geográficas, usando puntos, líneas y polígonos. Se suelen utilizar para representar calles o rutas, mediante líneas, así como ubicaciones mediante el uso de puntos. Es común el uso combinado de ambas, usando ráster para generar una capa base a partir de imágenes satelitales y agregando una capa vectorial para mostrar las calles por ejemplo.

Shapefile Se trata de un formato de archivos creado por la empresa ESRI para el almacenamiento de información geográfica. Es el formato más usado para el intercambio de este tipo de información, ya que permite almacenar puntos, multipuntos (multiples puntos), polilíneas (múltiples líneas) y polígonos. De todos modos, cada shapefile permite almacenar un único tipo de entidad, por lo que se puede requerir más de un Shapefile si se quieren almacenar varios tipos de figuras. También es importante recalcar que la in-

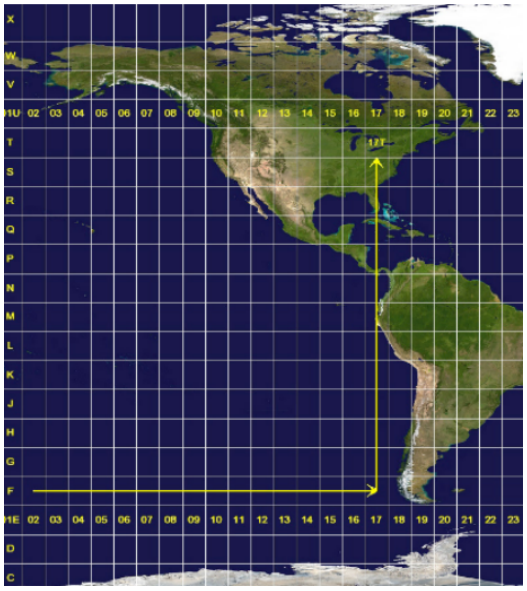


Figura 3.2: Sistema UTM [1].

formación no se guarda en un único archivo Shapefile, sino que se necesitan cuatro archivos diferentes: archivo principal (.shp), archivo de índice (.shx), archivo DBase (.dbf) y archivo de proyección (.prj).

Open Geospatial Consortium (OGC) El Open Geospatial Consortium (OGC) es un consorcio internacional que tiene como metas proveer estándares abiertos y públicos para el manejo de información geográfica. Es importante remarcarlo, ya que la mayoría de los servicios geográficos se basan en estos estándares.

3.2. PostgreSQL - PostGIS

A pesar de la existencia de los Shapefiles que sirven para almacenar la información geográfica, es deseable contar con algún mecanismo centralizado para el almacenamiento de este tipo de datos. La primer gran alternativa consiste en contar con bases de datos que nos permitan almacenar la información geográfica.

La OGC ha definido el Simple Features Standard (SFS) [13], que es un modelo de datos para objetos espaciales, el cual consta de dos parte fundamentales. Por un lado se cuenta con una arquitectura que define el modelo de objetos, y por otro lado tiene una implementación en SQL, la cual da la posibilidad de construir bases de datos geográficas.

En base a este estándar es que fue implementado PostGIS como complemento de PostgreSQL, que transforma una base de datos de ese tipo (PostgreSQL) en una base de datos geográfica [14]. PostgreSQL es un sistema de gestión de bases de datos relacional orientado a objetos, con licencia de libre utilización, motivo por el cual es ampliamente utilizado.

3.3. Geoserver

GeoServer es un servidor de código abierto escrito en Java, que permite el manejo de datos geográficos de forma simple. Implementa los principales protocolos y estándares planteados por la OGC, lo que garantiza su interoperabilidad, dando gran flexibilidad a la hora de crear mapas y compartir información y datos geográficos [15].

Entre las funcionalidades ofrecidas por Geoserver, se puede destacar un panel de configuración web. A través de esta interfaz es posible configurar la totalidad de las funcionalidades provistas por el servidor. Con ella se puede acceder fácilmente a las diferentes opciones de seguridad, configurar los datos que se expondrán, configurar diferentes fuentes de datos, realizar configuraciones de diseño, entre otras opciones.

Algunas de las opciones que se destacan como fuente de datos, son el uso de Shapefiles, y la conectividad con bases de datos PostGIS [16]. A partir de estas fuentes de datos se pueden definir las diferentes capas que formarán parte de las soluciones que se deseen implementar. Es así que se puede crear una capa por cada tabla geográfica de la base de datos PostGIS que se tenga, y luego combinarlas con alguna herramienta en una capa superior, para generar un mapa con los datos que se deseen mostrar.

Como protocolos de salida para exponer los datos se destacan WFS (Web Feature Service), WCS (Web Coverage Service) y WMS (Web Map Service) [17]. Éstos tres estándares de la OGC permiten exponer las capas a través de diferentes formatos:

Web Map Service (WMS) Este protocolo es utilizado para la transferencia de mapas en formato de imágenes generados a partir de información geográfica [18].

Web Feature Service (WFS) El protocolo WFS permite la transferencia de diferentes características de la información geográfica a la que accede el servidor, como ser atributos de los objetos [19].

Web Coverage Service (WCS) Este protocolo permite trasmitir capas raster [20].

Además del manejo de datos Geoserver también permite la personalización de los estilos de salida, mediante el uso de SLD (Styled Layer Descriptor) [17] [21]. El SLD es un esquema XML propuesto por el OGC como estándar para definir la apariencia de las diferentes capas que forman parte de un mapa. Este esquema permite definir el estilo visual de cada capa de objetos geográficos que componen el mapa permitiendo, por ejemplo, representar el color de relleno, tipo y ancho de borde de un punto, entre otras características.

3.4. OpenLayers

OpenLayers es una biblioteca de JavaScript con la cual se pueden construir visualizadores web de mapas e información geográfica [22]. Se trata de una biblioteca de código abierto, por lo que puede ser usada libremente.

Como OpenLayers está basado en Javascript, está fuertemente asociado al diseño orientado a objetos, donde, por ejemplo, cada mapa es considerado como un objeto. También se destaca por dar soporte a múltiples formatos y aceptar varias fuentes de datos, destacándose especialmente los protocolos de la OGC como son WMS y WFS.

Una característica a remarcar es que OpenLayers cuenta con una documentación extensa, en la que se destacan los ejemplos que plantea para las diferentes funcionalidades. Además cuenta con una guía de inicio rápida, con la cual se puede colocar un mapa sencillo en un sitio web en cuestión de minutos.

3.4.1. Conceptos básicos

A continuación se plantean algunos conceptos básicos sobre el funcionamiento de OpenLayers [23].

Map El concepto de Map (o mapa) es el componente central de OpenLayers. Usualmente se utiliza un elemento div (html) en el cual se renderiza el mapa. Las diferentes propiedades de un Map pueden ser configuradas al momento de construirlo o ser modificadas con algunos de los métodos provistos por la biblioteca.

View El objeto Map no es el encargado de realizar las reproyecciones al hacer zoom o al moverse por el mapa, ni de ubicar la posición del mapa que se

desea mostrar. De esto se encarga el objeto View (la vista). A cada mapa se le debe asignar una vista que será la que tendrá la proyección con el sistema de coordenadas que se deba usar, así como la resolución del mapa. También se encargará del manejo del zoom en caso que se desee usar esta opción.

Source El objeto Source se encarga de resolver el acceso a las diferentes fuentes de datos remotas. A través de este objeto se pueden configurar los servicios WMS o WFS que se necesiten, o también utilizar algunas fuentes de datos libres como puede ser OpenStreetMap.

Layer Cada Layer (capa) es una representación visual de los datos de las diferentes Sources. OpenLayers maneja diferentes tipos de Layers, entre las que se detacan las imágenes recibidas a través de web services geográficos. Cada mapa puede estar compuesto por varias capas, las cuales se superponen para mostrar la información deseada.

3.5. QGIS

QGIS es un Sistema de Información Geográfica (SIG) de código abierto. Se ejecuta como una aplicación de escritorio sobre Windows, aunque también hay versiones para Linux, Unix, Mac OSX y Android. Soporta numerosos formatos y funcionalidades de datos vector, datos ráster y bases de datos geográficas.

Algunas de sus principales características, de acuerdo a la documentación oficial [24], son la visualización de datos y las posibilidades que brinda para explorar datos y componer mapas.

QGIS es una herramienta que permite visualizar diferentes capas de diferentes formatos, como ser capas vectoriales y ráster. Algunos de los formatos de datos admitidos son las tablas y vistas habilitadas para operaciones espaciales utilizando PostGIS y Oracle Spatial. También se aceptan archivos shapefile de ESRI. Otro formato aceptado por QGIS son los datos en línea, que se reciben mediante el uso de webservices de OGC, como pueden ser WMS, WCS y WFS.

Otra característica importante de QGIS es la composición de mapas. A través de una interfaz amigable, se pueden superponer diferentes capas para crear mapas y explorar datos geográficos en forma interactiva.

Algunas de las herramientas que se tienen a disposición incluyen:

- Navegador QGIS
- Gestor de base de datos

- Diseñador de mapas
- Editar/ver/buscar atributos
- Apoyo para guardar y restaurar proyectos

QGIS también permite crear, editar, administrar y exportar capas vectoriales y ráster en varios formatos. La herramienta permite la creación y edición de archivos shapefiles, así como la creación de tablas de base de datos espaciales desde archivos shapefiles utilizando el complemento de Administración de Bases de Datos. Otra funcionalidad interesante es que QGIS permite crear archivos shapefiles o tablas en bases de datos, a partir de archivos de texto con valores separados por comas (CSV por su sigla en inglés) con coordenadas de puntos.

Capítulo 4

Descripción del problema

En esta sección se presenta en forma detallada el problema que motiva este proyecto, así como algunos trabajos relacionados a la misma problemática.

4.1. Sistema de Transporte Metropolitano

En la ciudad de Montevideo, Uruguay, la intendencia viene llevando a cabo un cambio en la forma de organización del transporte, procurando crear un sistema integral y unificado de todo el transporte de la cuidad. Este se llama "Sistema de transporte metropolitano", conocido por sus siglas STM. El principal cometido de este sistema es lograr mejorar la movilidad urbana de los usuarios, es decir los pasajeros. Con esto se busca alcanzar una sociedad más justa e integrada, tomando en cuenta las necesidades de todos sus integrantes [25]. Actualmente se encuentran dentro de este sistema las principales empresas de ómnibus de la capital (Coetc, Comesa, Cutcsa, Ucot), con todos los ómnibus, líneas, paradas y destinos que integran el sistema.

Uno de los principales cambios que introdujo este nuevo sistema es la utilización de tecnología en las unidades de transporte. Todas las unidades cuentan con un sistema electrónico de expedición de boletos, el cual admite el uso de tarjetas electromagnéticas. Éstas últimas son las llamadas tarjetas STM, las cuales pueden ser utilizadas para pagar boletos o utilizar las diferentes modalidades de viaje existentes. Dichas tarjetas son personales, lo cual aporta información sobre la movilidad de cada uno de los individuos que utiliza el sistema.

Otra característica tecnológica que se incorporó a las diferentes unidades del transporte es un dispositivo de GPS, el cual realiza mediciones y las registra. Algunos de los datos que se guardan son la posición (coordenadas en las cuales se realiza la medición) y la velocidad instantánea. Profundizaremos

en estos aspectos más adelante, ya que se trata de unos de los puntos más importantes de este proyecto.

El sistema también incluye cambios en la infraestructura de la ciudad, tales como terminales de trasbordo (terminal Colón por ejemplo), corredores exclusivos para ómnibus (el corredor Garzón fue el primero) y sectores de calles por las cuales solo pueden circular los ómnibus (sendas "sólo bus"). Algunos de estos cambios han sido foco de críticas por parte de los usuarios y los medios de prensa se han hecho eco. Las obras más cuestionadas fueron la terminal Colón y el corredor Garzón, las cuales causaron molestias y demoras en los usuarios que transitaban por ellas.

4.2. Trabajos relacionados

Las autoras Joana Maria Seguí Pons y Maria Rosa Martínez Reynés abordan en su artículo "Los sistemas inteligentes de transporte y sus efectos en la movilidad urbana e interurbana"[26], algunas problemáticas del transporte, especialmente en las grandes ciudades. Algunos de los problemas que mencionan las autoras son el aumento de emisiones de dióxido de carbono a la atmósfera, y el aumento en las demoras e incumplimiento de horarios del transporte público. Para hacer frente a estos problemas, se creo la iniciativa "Civitas", que se desarrolla en 19 ciudades europeas, y cuyo principal objetivo es dar apoyo al desarrollo e implementación de medidas innovadoras y eficaces para mejorar la problemática del transporte urbano, buscando mejorar la eficacia y eficiencia del transporte y proveer seguridad a los usuarios. En ese contexto, los ITS *"constituyen elementos fundamentales en sus nuevas estrategias de gestión del transporte urbano e interurbano"*.

Los sistemas de transporte inteligentes, como indica el artículo, requieren de una combinación de información, comunicaciones y tecnologías del transporte en vehículos e infraestructuras. Los Global Position Systems (GPS) resultan sumamente importantes a la hora de generar datos, aportando gran cantidad de información. Según destacan las autoras, las aplicaciones que dan soporte a los ITS proveen asimismo eficiencia y seguridad en los desplazamientos. Ayudan a la detección y prevención de incidentes, y también pueden aportar soluciones para descongestionar las vías. Otra ventaja del uso de los ITS es la automatización de funciones y el acceso a la información, lo cual sirve para la optimización de los viajes, reduciendo los tiempos de los usuarios.

Aplicado a la red de transporte público, los ITS proporcionan los medios necesarios para conocer, regular y gestionar en tiempo real el funcionamiento y los recursos disponibles. También se busca, según las autoras, facilitar la

información necesaria para que los responsables y usuarios del transporte puedan tomar decisiones a fin de optimizar y mejorar el servicio. Este último punto es abordado por el presente proyecto, con el fin de dar valor agregado a los datos recabados, buscando generar una fuente de información útil para la toma de decisiones.

Finalmente las autoras plantean un escenario en el cual se transmite la información desde las unidades de transporte, a través de una computadora abordo, a una computadora central de una empresa de transporte, desde donde se procede al análisis de los datos y toma de decisiones del sistema de transporte. Los ejemplos que plantea son, el uso de los datos para evitar que se agrupen vehículos de una misma línea o que se separen demasiado, evitando de esa forma los problemas de demanda de viajeros (vehículos semivacíos o aglomeraciones innecesarias).

En el artículo "Intelligent Transportation Systems"[27], los autores Richard Weiland y Lara Baughman Purser, presentan algunos otros aspectos importantes sobre los sistemas de transporte inteligente. En primera instancia se plantean puntos fundamentales para el éxito en el uso de las tecnologías aplicadas al transporte, destacándose la importancia de atender a las preocupaciones sociales e institucionales, en busca de nuevas formas de negocio.

Los aspectos positivos que destacan los autores del uso de este tipo de sistemas radica fundamentalmente en la seguridad y movilidad que se brinda a los usuarios, y la reducción del uso de combustibles, logrando reducción de costos y de emisiones de gases a la atmósfera. Sin embargo consideran que se debe tener en cuenta, para lograr el éxito del sistema de transporte, que existen problemáticas que van más allá del sistema propiamente dicho. Se menciona por ejemplo los factores humanos y de seguridad, así como los problemas legales que puedan surgir. Según el artículo, cada vez son más comunes este tipo de sistemas para el transporte terrestre, sin embargo siempre debe ser claro el beneficio que puedan tener los consumidores, de modo que los costos que pueda implicar estén claramente justificados.

Terminando el artículo, los autores destacan que los sistemas de transporte inteligentes presentan oportunidades para la investigación, y que se debe tener en cuenta en todo momento que lo importante es mejorar la calidad del servicio, haciéndolo más eficiente, eficaz y seguro. Es por eso que el proyecto se aborda desde la óptica de buscar dar información que sirva para mejorar el STM, mejorando el servicio y la satisfacción de los usuarios.

Otro trabajo analizado fue "Innovaciones tecnológicas aplicadas al transporte colectivo en Quito"[28] de los autores Florent Demorae, Francis Bondoux, Marc Souris e Hidalgo Núñez. En este caso se aborda la problemática de un caso puntual, el sistema de transporte de la ciudad de Quito. El proyecto busca generar y analizar datos sobre el transporte, con el objetivo de

lograr mejoras en el servicio.

El primer problema que se presenta es la escasez de datos confiables y actualizados, con los cuales evaluar la demanda del transporte colectivo y llevar a cabo la fiscalización del mismo. Se busca calibrar la oferta necesaria de transporte, así como las frecuencias horarias y los recorridos. También se desea comprobar si las empresas cumplen con los permisos de operación.

Para la obtención de datos se presenta el uso de un sistema semiautomático llamado FINDEM, el cual permite generar datos del recorrido y de las velocidades en forma automática a partir de GPS, y realizar manualmente el conteo de pasajeros por parte de un encuestador. Con esos datos se puede determinar el recorrido de los vehículos, y determinar con precisión, dónde, cuándo, y cuántos pasajeros suben y bajan de los buses.

Para el procesamiento de los datos generados por el sistema FINDEM, se siguen una serie de etapas. En la etapa 1 se transfieren los datos a Excel, donde son depurados y luego integrados a una base de datos. Luego, en la etapa 2, se toman tramos de referencia relativamente cortos, para asignar información geográfica a los datos obtenidos de la etapa 1, en base a dichos tramos. Esta asignación se realiza en la etapa 3, fundamentado en que la señal satelital (GPS) suele distorsionarse levemente (por el número de satélites presentes en el cielo), además del hecho de que el mapa vial que sirve de referencia tiene un margen de error. Finalmente, en la etapa 4, se procede a la expansión de la muestra, multiplicando simplemente los datos iniciales obtenidos durante el conteo de un viaje, georreferenciados en los tramos, por el número de buses que salieron en la hora correspondiente.

A partir del trabajo, los autores logran obtener información valiosa, como ser la demanda total y la determinación de horas pico. Sobre esto último se destacan dos observaciones: la primera es que existen dos horarios picos, y que la demanda en dichos horarios no es coincidente en cuanto al sentido de movilidad de los pasajeros.

Uno de los puntos destacados de el trabajo es que los autores logran la generación y procesamiento de una gran cantidad de datos, obteniendo información sintética georreferenciada de gran interés para la toma de decisiones, siendo estos los mismos objetivos que presenta el proyecto.

Este último trabajo analizado tiene grandes similitudes y algunas diferencias con el proyecto, las cuales son aprovechadas y contrastadas con la realidad del STM. La primer gran diferencia consiste en la fuente de datos, ya que el STM provee grandes volúmenes que se generan constantemente. De todos modos ambos proyectos manejan datos georreferenciados, los cuales pueden sufrir distorsiones que deben ser consideradas al momento de trabajar con este tipo de datos. Otro punto en común entre el trabajo y el proyecto es la metodología en etapas, haciendo una depuración o preprocesamiento de

los datos de entrada. Otra similitud entre el trabajo y el proyecto se puede considerar el intentar determinar horas pico. También se destaca que en ambas propuestas se trabaja manejando datos a nivel de tramos relativamente cortos.

4.3. Motivación del proyecto

Como se vio anteriormente, el STM genera una gran cantidad de datos, los cuales se pueden utilizar de diferentes formas. En la actualidad la Facultad de Ingeniería ha logrado acceder al conjunto de datos de los GPS de los ómnibus, a través de los cuales se puede obtener varias características de los viajes, como ser la velocidad o el tiempo real del recorrido. Resulta de especial interés el procesamiento de este conjunto de datos para un mejor aprovechamiento de los mismos, lo cual impone un desafío debido a la gran cantidad de datos que se tienen.

Por otro lado, es necesario que los datos que se procesan puedan ser presentados a los usuarios, en especial a aquellos actores que participen en la toma de decisiones. Para ello se observa como una oportunidad el uso de herramientas geográficas, ya que los datos de las líneas, en especial sus recorridos, al igual que las posiciones de las paradas, se encuentran georreferenciados. Es así que el proyecto busca mostrar los datos mediante el uso de mapas, a través de una aplicación web, la cual puede ser accedida mediante dispositivos móviles.

La principal motivación de este proyecto es generar una herramienta eficiente para el procesamiento de los datos, generando información valiosa para la mejora del servicio, con el objetivo de lograr una mejor experiencia por parte de los usuarios finales, es decir de los pasajeros que hacen uso día a día del STM.

4.4. Datos a procesar

En este proyecto se toma como punto de partida los datos de los GPS de las unidades de transporte que conforman el STM. En particular, la Intendencia de Montevideo (IM) concedió a la Facultad de Ingeniería los datos que disponía de todos los viajes realizados por los ómnibus durante el año 2015. Los mismos vienen dados en archivos CSV. Estos archivos están divididos por meses, hay uno por cada mes del año 2015 y tienen un tamaño de aproximadamente 10GB cada uno.

Dentro de cada uno de estos archivos se encuentran los registros realizados

por los dispositivos GPS de cada uno de los ómnibus, en cada viaje realizado durante el mes correspondiente. En la Tabla (4.1) se muestran los campos que componen cada registro, es decir las columnas que forman el archivo.

Columna	Explicación
Línea	Indica la línea de ómnibus a la cual pertenece el viaje
Timestamp	Fecha y hora en que se tomó la medición
coordenada Lat	Latitud de la coordenada en que se realizó la medición
coordenada Lon	Longitud de la coordenada en que se realizó la medición
Id viaje	código identificador del viaje
Velocidad	Velocidad instantánea al momento de realizar la medición
Cod Variante	Código de la variante de línea a la cual corresponde el viaje
es Parada	Indica si se trata de una medición en una parada(7) o no(6)
Cod parada	En caso que corresponda, indica el código de la parada

Tabla 4.1: Campos de archivos de entrada (mediciones GPS).

A partir de estos datos es posible reconstruir cada viaje, viendo los tiempos de demora entre cada parada, y luego calcular la velocidad promedio.

Uno de los objetivos que presenta este proyecto es el cálculo de las velocidades promedio de los viajes, agrupándolos por franjas horarias y por variantes de líneas. Un punto que resulta importante aclarar es que las agrupaciones se hacen por variantes de líneas y no por líneas. Para ello se debe entender el concepto de variante de línea y distinguirlo de la línea. Una línea de ómnibus es aquella que une dos puntos de la ciudad realizando un determinado recorrido. Por ejemplo podemos tomar la línea 174 que une Punta Carretas con Aviación. Ésta línea tiene inicialmente dos recorridos: una que comienza en Punta carretas y finaliza en Aviación, y otro que hace el recorrido en el sentido opuesto. Cada uno de estos recorridos puede ser considerado como una variante de la línea. Además pueden existir destinos intermedios, como por ejemplo, la línea 174 tiene un recorrido que une Punta Carretas con Peñarol. En este caso tendríamos otra variante de línea. Con estos ejemplos se pueden ver fácilmente el significado y las diferencias entre línea y variante de línea.

En el catálogo de datos abiertos del estado [29] se encuentran disponibles los datos de las líneas de ómnibus, las cuales se encuentran separadas por variantes de línea (ver Tabla 4.2). A partir de estos datos se puede obtener una lista de todas las variantes de líneas que forman parte del STM. También forma parte de este conjunto de datos un archivo de formato Shapefile (con sus archivos complementarios), en el cual se tienen los recorridos de todas

esas variantes de líneas, trazados como un conjunto de líneas (polilínea).

Nombre	Descripción
Gid	Código identificador de uso interno
cod_linea	Código de línea
desc_linea	Descripción de línea de transporte (ej: 145, D10, etc)
ordinal_sublinea	Número correlativo de la sublinea en la línea
cod_sublinea	Código de la sublínea de recorrido
desc_sublinea	Descripción, origen y destino
cod_variante	Código de la variante de recorrido (vincula con paradas)
desc_variante	Descripción variante

Tabla 4.2: Campos de archivo de líneas.

Otro conjunto de datos disponible en el catálogo de datos abiertos es el de las paradas del STM, contando con sus ubicaciones y algunos otros datos, como ser qué variantes de líneas pasan por dichas paradas (ver Tabla 4.3). Al igual que como ocurre con los recorridos de las líneas, se cuenta con un archivo Shapefile (y sus archivos complementarios) donde se tienen las paradas ubicadas, representadas como puntos, obteniéndose un conjunto que representa todas las paradas del STM.

Columna	Explicación
Cod_ubic_p	Código de la ubicación de parada
Cod_varian	Código de la variante de línea de ómnibus
Ordinal	Número correlativo de la parada en el trayecto de la variante
Calle	Nombre de la calle sobre la que se ubica la parada
Cod_calle1	Código de la Calle según nomenclator oficial de Montevideo
Esquina	Nombre de la esquina más próxima
Cod_calle1	Código de la Esquina según nomenclator oficial de Montevideo
X	Coordenada X de la ubicación (SIRGAS2000 UTM 21s)
Y	Coordenada Y de la ubicación (SIRGAS2000 UTM 21s)

Tabla 4.3: Campos de archivos de entrada (Paradas).

4.5. Visualización de resultados

Como se planteó en la sección de motivación del proyecto, es necesario contar con una interfaz para exponer los datos resultantes del procesamiento. Al tratarse de conjuntos de datos con información geográfica, es posible exponerlos utilizando herramientas para este tipo de datos, mostrando los resultados en un mapa.

Un desafío que se presenta es el del almacenamiento de los datos junto a su información geográfica. Aquí surgen varias alternativas, como pueden ser una base de datos geográfica o una serie de archivos, como los Shapefiles. En el caso de este proyecto se opta por el uso de una base de datos geográfica.

Luego se debe planificar la forma en que los datos serán expuestos, donde existen varias posibilidades, dependiendo del uso que se les quiera dar. Si se quieren exponer mediante webservices, es necesario contar con soporte para webservices geográficos como WMS o WFS, los cuales deben ser manejados mediante un servidor geográfico como puede ser Geoserver. La opción elegida en este proyecto es el uso de WMS combinado con WFS.

Finalmente se debe decidir cómo se mostrarán los mapas y los datos en ellos. Se puede trabajar con un cliente de escritorio como QGIS o crear un sitio web utilizando alguna herramienta de visualización de mapas, como OpenLayers. También se debe resolver el diseño y presentación de la información. Esto se aborda con mayor detenimiento en los siguientes capítulos.

Capítulo 5

Arquitectura y diseño de la solución

El presente proyecto se encuentra particionado en dos secciones parcialmente independientes: el procesamiento de los datos de entrada y la visualización de los resultados. A pesar de la independencia del desarrollo de ambas partes, se hace necesaria una coordinación entre ambas, ya que la salida del procesamiento de datos será la entrada principal para la parte de visualización de resultados.

5.1. Análisis preliminar

Conociendo los datos de entrada con los que se cuenta, se plantea la posibilidad de realizar el cálculo de la velocidad promedio, en cada mes, para cada una de las variantes de línea que forman parte del STM. Además, para aumentar la granularidad de la información, se considera necesario que no se calcule una única velocidad en promedio para la totalidad del recorrido, sino que en cambio se puedan definir tramos y calcular la velocidad promedio en cada uno de ellos. Una partición que resulta natural para un recorrido de un ómnibus, es entre cada par de paradas que forman parte de su recorrido. Sabiendo que se cuenta con la información de las paradas, es que se tomó la decisión de utilizar estos tramos como otra partición del problema.

Es así que el procesamiento de los datos se realiza en forma particionada, para cada variante de línea, para cada mes, y a su vez se consideran diferentes tramos, los cuales van desde una parada del recorrido hasta la siguiente parada que forma parte del recorrido de la variante de línea que se analiza.

Al tener estas particiones, las cuales son independientes, se observa una posibilidad real de paralelizar el procesamiento de los datos, con el objetivo

de disminuir los tiempos de ejecución, en búsqueda de un mayor aprovechamiento de los recursos computacionales.

5.2. Procesamiento de datos

Para el procesamiento de datos, se ha decidido utilizar una estrategia de programación paralela multithreading, haciendo uso de la infraestructura de la facultad, mediante el uso de una computadora de alto rendimiento y gran poder de cómputo.

La arquitectura que se utiliza para el procesamiento de datos es una arquitectura en pipeline, la cual consta de tres etapas que se realizan en forma secuencial. Cada una de estas etapas genera insumos para la siguiente, obteniéndose el resultado al finalizar la última etapa del pipeline. Las tres etapas son las siguientes: preparación de datos, procesamiento principal y post-procesamiento.

5.2.1. Etapa 1: preparación de datos

En la primera etapa se realiza la preparación de los datos, realizando una partición de los mismos para su posterior procesamiento.

Por un lado se deben dividir las mediciones de las diferentes unidades de transporte, las cuales se presentan en un único archivo CSV ordenadas por el timestamp asignado a cada una de esas mediciones. En este trabajo la división se realiza en viajes, generando un nuevo archivo CSV por cada viaje. Esto es posible porque cada línea del archivo de entrada tiene un campo que es un identificador de viaje. También se realiza al mismo tiempo un log de viajes correspondientes a cada variante de línea.

La forma de realizar la división de los viajes es mediante la ejecución un script que crea un nuevo archivo CSV para cada uno de los viajes del mes que se analiza, nombrando a cada archivo con el identificador del viaje. Al mismo tiempo, y a través del mismo script, se crea un conjunto de archivos CSV por cada variante de línea. En estos archivos se almacenan los identificadores de los viajes correspondientes a cada variante de línea.

Esta parte del proceso que consiste en tomar un archivo y dividir su contenido en archivos más pequeños, se debe realizar en forma secuencial, ya que se deben recorrer la totalidad de sus registros.También se debe realizar la preparación de los tramos que se utilizarán para el cálculo de las velocidades, tiempos y distancias. Para esta otra parte se cuentan con los datos de las paradas correspondientes al recorrido de cada variante de línea, y los datos de las paradas.

A partir de los recorridos de las diferentes variantes de línea, los cuales tienen un ordinal que indica el orden de cada parada, se van generando los tramos de cada variante de línea. En este trabajo se optó por definir cada tramo como el trayecto que hay entre cada par de paradas contiguas. Así se tiene el tramo uno como aquel que va desde la parada uno a la parada dos, luego el tramo dos como aquel que va desde la parada dos hasta la parada tres del recorrido, y así sucesivamente hasta llegar a la última parada del recorrido.

El proceso de generación de los tramos se realiza con un único script que toma como entradas los recorridos de las variantes de líneas y los datos de las paradas. A partir de este proceso se genera un archivo CSV por cada variante de línea conteniendo una línea con cada tramo. En la Tabla 5.1 se listan los campos que conforman cada línea del archivo de salida.

Columna	Explicación
orden	número que indica el orden del tramo en el recorrido
cod_parada_ini	Código de la parada inicial del tramo
cod_parada_fin	Código de la parada final del tramo
distancia	Distancia entre las paradas en metros
xIni	Coordenada x de la parada inicial del tramo (SIRGAS2000 UTM 21s)
yIni	Coordenada y de la parada inicial del tramo (SIRGAS2000 UTM 21s)
xFin	Coordenada x de la parada final del tramo (SIRGAS2000 UTM 21s)
yFin	Coordenada y de la parada final del tramo (SIRGAS2000 UTM 21s)

Tabla 5.1: Campos de tramos

Los datos de las paradas son utilizados para definir las coordenadas de comienzo y fin de cada tramo, que coinciden con las coordenadas de las paradas que definen el tramo en cuestión. A partir de las coordenadas también se realiza el cálculo de la distancia total del tramo, la cual se utilizará en la etapa del procesamiento principal para calcular la velocidad de cada variante de línea en cada tramo.

5.2.2. Etapa 2: procesamiento principal

La segunda etapa es la principal del trabajo, ya que es en ella donde se realizan las búsquedas de los tiempos de pasada por los diferentes puntos y los cálculos sobre las velocidades en cada tramo.

Para calcular las velocidades en cada tramo, es necesario hallar previamente los tiempos por los que las unidades de transporte pasan por los di-

ferentes puntos de los recorridos, en este caso las paradas que definen cada tramo.

Se toman como insumos de entrada de esta etapa del procesamiento los datos obtenidos en la parte anterior: los viajes de cada variante de línea, los registros de cada viaje y los tramos definidos para cada variante de línea. A partir de estos datos de entrada se realiza el cálculo de las demoras promedio de cada variante de línea en cada tramo, para luego definir la velocidad promedio en dicho tramo.

A partir de la observación de que cada variante de línea es independiente de las demás, se puede particionar intuitivamente el problema, permitiendo la resolución en forma paralela para cada una de las variantes de línea. La estrategia que se usa en este caso es el uso de programación mutithreading, asignando un hilo de ejecución a cada variante de línea. De esta forma cada hilo de ejecución se encarga de analizar los diferentes viajes de la variante de línea que tiene asignada.

Un punto a destacar en esta etapa es que los viajes ya se encuentran divididos, garantizando que los diferentes hilos de ejecución no compiten por el uso de los diferentes archivos de entrada. Lo mismo ocurre con los archivos que contienen los tramos los cuales también se encuentran divididos por variantes de línea.

Como resultado de esta etapa se genera un archivo CSV por cada variante de línea, conteniendo una línea de texto por cada tramo. Las columnas que componen el archivo de salida se basan en los datos de los tramos (obtenidos en la etapa anterior). A esas columnas se le agregan algunas nuevas como ser el tiempo promedio, la velocidad promedio y la cantidad de mediciones consideradas para el cálculo.

En la Tabla 5.2 se listan los campos que conforman cada línea del archivo de salida del procesamiento principal.

5.2.3. Etapa 3: post-procesamiento

La etapa anterior del procesamiento de datos finaliza con un conjunto de archivos de salida separados por variantes de línea y por mes. En esta última etapa de post-procesamiento se procede a “juntar”los datos obtenidos anteriormente, de forma de generar un único archivo CSV en el cuál se tienen la totalidad de los datos obtenidos para el año que se esta procesando.

La estrategia consiste en usar un algoritmo secuencial, al igual que en la primera etapa, el cual recorre los archivos de cada variante de línea y cada mes del año que se está procesando, generando un único archivo de salida para cada año analizado.

Columna	Explicación
orden	número que indica el orden del tramo en el recorrido
cod_parada_ini	Código de la parada inicial del tramo
cod_parada_fin	Código de la parada final del tramo
Distancia	Distancia entre las paradas en metros
Tiempo	Tiempo promedio para el tramo en segundos
Velocidad	Velocidad promedio para el tramo en kilómetros por hora
Cant_mediciones	Cantidad de mediciones válidas utilizadas para el cálculo
xIni	Coordenada x de la parada inicial del tramo (SIRGAS2000 UTM 21s)
yIni	Coordenada y de la parada inicial del tramo (SIRGAS2000 UTM 21s)
xFin	Coordenada x de la parada final del tramo (SIRGAS2000 UTM 21s)
yFin	Coordenada y de la parada final del tramo (SIRGAS2000 UTM 21s)

Tabla 5.2: Campos de tramos de archivos de salida

5.3. Uso de QGIS

Dentro de los datos obtenidos en la etapa de procesamiento, se tiene información geográfica pero no en el formato usual (como una geometría) sino como texto. Puntualmente se cuenta con las coordenadas de las diferentes paradas que definen los tramos de cada variante de línea.

A través del uso de la herramienta QGIS se puede crear geometrías desde los datos de las coordenadas almacenados en archivos de texto como ocurre en este caso. Es así que utilizando esta funcionalidad de QGIS se crean las diferentes geometrías a partir del archivo CSV que se obtiene como resultado de la etapa de procesamiento de datos. Estas geometrías pueden ser almacenadas somo archivos Shapefiles o en alguna base de datos geográfica. En este proyecto se optó por la segunda opción como se verá en la siguiente sección.

5.4. visualización de resultados

Para la visualización de los resultados se plantea el uso se una arquitectura en capas, y un despliegue en varios servidores distribuidos. Ésta estrategia tiene como ventaja la independencia de los diferentes componentes, facilitando futuras ampliaciones o trabajos relacionados ya sea compartiendo datos, ampliando los servicios o generando nuevas herramientas de visualización de los datos.

En el diagrama 5.1 se puede observar los componentes de la arquitectura

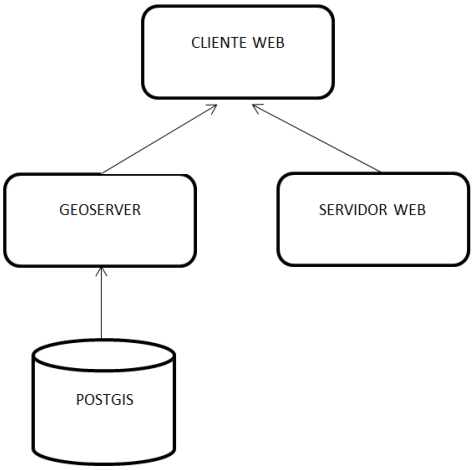


Figura 5.1: Arquitectura.

5.4.1. Base de Datos

Por el lado del almacenamiento de los datos se plantea el uso de una base de datos geográfica, dado que se cuenta con datos georreferenciados ya convertidos en geometrías. Esta base de datos es la que contiene los datos de los tramos de cada variante de línea definidos por la ubicación geográfica de las paradas de cada tramo. De esta forma se conforma la capa inferior de la arquitectura de este proyecto.

La principal motivación para el uso de una base de datos es que la cantidad de datos que se maneja es alta. Al usar una base de datos, el mantenimiento del sistema es más sencillo, y además se facilita la forma de compartir los datos generados por este proyecto.

La base de datos elegida es PostgreSQL, la cual es de licencia libre. La elección de ésta tecnología se da por varios factores. En primer lugar se tuvo en consideración que se trata de una base de datos que se puede utilizar gratuitamente, lo cuál reduce los costos del proyecto. Otro factor importante es que se trata de una base de datos ampliamente extendida en el mercado y con una documentación extensa. Finalmente se tomo la determinación del uso de esta base de datos por los complementos con los que se cuenta, que permiten transformarla en una base de datos geográfica.

Para el manejo de datos geográficos se utiliza el complemento PostGIS. Éste complemento es el que se encarga del manejo de datos geográficos a nivel de la base de datos, permitiendo almacenar las geometrías definidas por el estándar definido por la OGC.

5.4.2. Servidores

La segunda capa de la arquitectura de éste proyecto es la capa de negocio. En esta capa encontramos dos servidores por separado: un servidor web y un servidor geográfico. El primero de los servidores se encarga del sitio web de acceso a los datos por parte de los clientes, mientras que el segundo servidor se encarga de los datos geográficos, exponiendo dichos datos mediante webservices geográficos.

En el caso de este proyecto se cuenta con una capa de negocio fina, ya que la lógica desarrollada en ésta capa es poca. Esta capa de negocio tiene dos tareas principales, y cada una es llevada a cabo por cada uno de los dos servidores.

El primero de los servidores es un servidor web, el cual atiende las solicitudes de los clientes exponiendo el sitio de visualización de los resultados del proyecto. El segundo de los servidores es el servidor geográfico, que tiene como tarea principal la transmisión de los datos geográficos, mediante el uso de webservices, desde la base de datos al cliente que los solicita.

Para servidor geográfico se optó por el uso de Geoserver, ya que es el más extendido en el mercado. Se trata de un servidor estable y confiable, con una extensa documentación y una comunidad activa para plantear y evacuar dudas. También se trata de un herramienta de licencia de uso libre, lo cual permite su uso gratuito.

5.4.3. Capa de presentación

En la capa superior de la arquitectura se utiliza un sitio web para la visualización de los datos. Éste sitio expone los resultados del proyecto mediante el uso de un mapa de la ciudad de Montevideo, sobre el cual se muestran las líneas de ómnibus, sus paradas y las velocidades del tramo de llegada a cada una de ellas para cada variante de línea.

En esta capa se implementa un cliente grueso, ya que posee gran parte de la lógica. La programación del cliente web se hace mediante el uso de javascript utilizando la biblioteca OpenLayers. Dicha biblioteca es la encargada de resolver el manejo de los datos geográficos, desde las solicitudes que se hacen al servidor geográfico, hasta la forma de mostrar los datos que se reciben a través de los webservices expuestos por dicho servidor.

Capítulo 6

Implementación

En esta sección se plantean los detalles de la implementación de las diferentes etapas del proyecto. Se destaca que se siguen los lineamientos planteados en la etapa de análisis y diseño de la arquitectura de la solución.

6.1. Procesamiento de datos

Para el procesamiento de datos se utilizaron dos lenguajes de programación diferentes. Por un lado, para la separación de los viajes, se utiliza un script en Python, mientras que para las otras etapas se utilizan programas implementados en Scala.

6.1.1. Etapa 1

En la etapa de preparación de datos se cuenta con dos programas. El primeo de ellos se encarga de la separación de los viajes, mientras que el segundo se encarga de la generación de los tramos de cada variante de línea.

6.1.1.1. Separación de viajes

Inicialmente se cuenta con un archivo CSV por cada mes, el cual cuenta con todos los registros de todos los viajes realizados en el mes por las diferentes unidades del sistema de transporte colectivo de la ciudad de Montevideo. Estos archivos tienen millones de líneas de texto y un volumen superior a los 10GB. Esas dimensiones hacen que su manejo sea complejo, por lo que se realiza la división de cada uno de los archivos.

La división que se plantea en este proyecto es por viajes. Esto es posible porque cada viaje tiene un identificador numérico único, por lo que se pue-

den seleccionar solamente las líneas del archivo que coincidan en el código identificador del viaje.

Esta etapa del proyecto se realiza mediante un programa escrito en Phyton que recorre cada linea de texto de cada uno de los archivos mensuales. De cada una de esas líneas se analizan dos datos: el identificador del viaje y la variante de línea a la que pertenece. Este análisis se puede hacer fácilmente por la estructura del texto, ya que se trata de un archivo CSV por lo que las columnas de datos están separadas por comas.

Cada línea que lee el proceso se reescribe en un nuevo archivo cuyo nombre coincide con el identificador del viaje, creando el archivo si aún no existe o agregándola al final del archivo si el mismo ya fue creado (se puede haber creado al encontrar un registro previo del viaje).

Otra tarea que se realiza al leer cada línea, es identificar la variante de línea a la que pertenece el viaje. Una vez identificado el viaje y la variante de línea, se escribe el identificador del viaje en un archivo que se asocia a cada variante de línea. Esta asociación se hace nombrando al archivo con el identificador de la variante de línea que es único. Al igual que sucede con los registros de los viajes, cuando se guarda el identificador en el archivo correspondiente a la variante de línea, se puede crear el archivo o escribir una línea al final de uno en caso que ya exista.

Dada la naturaleza del formato de los datos de entrada, y que se deben realizar varias lectura y escrituras en archivos, no es posible paralelizar este proceso. Sin embrago se debe considerar que esta preparación de archivos se realiza una única vez para cada archivo con datos de cada mes, lo cual disminuye el impacto de su demora.

6.1.1.2. Generación de tramos

El otro conjunto de datos que se debe preparar, son los tramos de las diferentes variantes de línea. Los tramos definidos para este proyecto son los que van entre cada par de paradas contiguas en el recorrido de cada variante de línea.

La información con la que se cuenta consiste en las diferentes variantes de líneas que forman parte del STM, los recorridos de cada una de esas variantes de línea, los cuales incluyen las paradas por la que pasan y el orden en que recorren esas paradas. También se cuenta con la información de las paradas, incluyendo sus coordenadas en el formato UTM.

A partir de esa información que viene dada en tres archivos CSV, uno con las variantes de línea, otro con los recorridos y otro con las paradas, se ejecuta un programa implementado con el lenguaje Scala que los analiza y crea los tramos de cada variante de línea.

El primer paso del proceso consiste en cargar en memoria los datos de los tres archivos, para lo cual se abren y recorren en su totalidad. Luego el proceso va recorriendo una lista de variantes de línea generada al leer el archivo correspondiente (el listado de las variantes de línea), y realiza la generación de los tramos para cada una de esas variantes de línea, mediante la función "generadorTramosPorLinea"que se encarga de todos los cálculos necesarios.

La función generadora de tramos comienza buscando la cantidad de paradas asociadas a la variante de línea que se está analizando. Ésto lo hace mediante la búsqueda del mayor ordinal entre las paradas de la variante de línea. Luego se crea una lista con las paradas de la variante de línea. Finalmente se procede a la creación de los tramos, los cuales se hacen en orden de menor a mayor, iniciando por el que va de la parada 1 a la 2, siguiendo por el tramo que va de la parada 2 a la 3 y así sucesivamente.

A medida que se genera un tramo, se obtienen las coordenadas de cada una de las paradas que lo definen. Estas coordenadas se agregan como datos del tramo, y además son utilizadas para el cálculo de la distancia entre las paradas, la cual se considera como la longitud del tramo. Se tomó la decisión de realizar ésta simplificación en el cálculo de la longitud de los tramos, el cual implica un pérdida de exactitud en algunos escenarios. Sin embargo, en la mayoría de los casos los tramos son en línea recta (en estos casos no hay error en el cálculo de la distancia), por lo que no se sufren grandes desviaciones en el cálculo de sus longitudes.

Para el cálculo de la distancia se toma en consideración un punto fundamental: las coordenadas de las paradas vienen dadas en formato UTM. Esto garantiza que, al tratarse de distancia cortas comprendidas en el mismo cuadrante, se puede aplicar el teorema de Pitágoras obteniendo como resultado la distancia en metros entre ambos puntos.

A medida que se van generando los tramos, éstos se van escribiendo en un archivo CSV cuyo nombre coincide con el código de la variante de línea. Al ser creados en forma ordenada, se garantiza que el archivo resultado tiene los tramos ordenados de igual forma.

6.1.2. Etapa 2

En la segunda etapa del procesamiento de datos es cuando se realizan los principales cálculos y se genera la información deseada. En el caso de este proyecto se busca hallar las demoras (en segundos) y velocidad promedio (en kilómetros por hora) para cada variante de línea en cada tramo, permitiendo distinguir algunas franjas horarias críticas.

Para este procesamiento se utiliza un programa en Scala el cual tiene

algunos parámetros de entrada.En primera instancia se debe definir los rangos horarios en los cuales se realizarán los cálculos. Se pregunta por la cantidad de tramos horarios se quieren considerar, y luego se pregunta por la hora de inicio y la hora de finalización de cada uno de esos tramos horarios. En caso de no definir ningún rango horario, se tendrán en consideración todas las mediciones, sin importar en que horario hayan sido tomadas.

Otro parámetro que se solicita a continuación y que debe ser ingresado por el usuario es el nivel de paralelismo que se usará en el proceso, es decir la cantidad de hilos de ejecución con los que se desea trabajar.

Luego se cargan las diferentes variantes de línea a partir del archivo de líneas. Esta carga se hace en una lista secuencial, en la cual se vuelcan los códigos de las diferentes variantes de línea. A partir de esa lista de variantes de línea, se utiliza la función "par"de Scala, que convierte a la lista secuencial en una lista paralela. Luego se procede a realizar los cálculos para cada variante de línea en forma paralela, llamando a la función "analisisPorLinea".

Las tareas de la función "analisisPorLinea", consisten en cargar los viajes de la variante de línea que se está analizando. Para obtener estos datos, busca el archivo generado en la etapa 1, identificado con el código de la variante de línea. El archivo de viajes de la variante de línea contiene los identificadores de los viajes que se deben analizar, ya que son los que corresponden a la variante de línea que se está procesando. También se cargan en esta etapa los tramos de la variante de línea para los cuales se realizarán los cálculos de tiempo y velocidades promedio.

Una vez que se cuenta con los viajes y los tramos cargados en memoria, se procede a cargar los registros de cada uno de esos viajes y a procesarlos para obtener los datos deseados. El procesamiento consiste en encontrar los dos registros del viaje que sean más cercanos (en distancia) a las dos paradas que definen cada tramo. Para esto se hace uso de las coordenadas de las paradas y de las coordenadas de las mediciones de los viajes. En este punto se deben hacer dos consideraciones: la primera es que se define una cota máxima para decir que una medición es lo suficientemente cercana a una parada y la segunda es que se debe realizar conversiones entre las coordenadas, ya que se encuentran en diferentes formatos.

Para que una medición sea considerada válida se debe encontrar a menos de treinta metros de la parada más cercana. Éste valor se desprende del análisis de algunos datos; por un lado se tiene que las mediciones son usualmente cada 15 segundos, observando los registros de los viajes. Por otro lado se obtuvo a partir del análisis de los horarios que deben cumplir los ómnibus, su velocidad debe ser en promedio de 15 km/h, lo que equivale a 4 m/s. Con estos datos se observa que la distancia entre dos medidas, en promedio, se da cada 60 metros, por lo que situando la parada en el punto medio, se obtienen

mediciones a los 30 metros. de todos modos las velocidades son variables y usualmente en las cercanías de las paradas las velocidades pueden ser menores, en cuyo caso existirán más de una medida válida para la parada en cuestión. en este caso el proceso se quedará con la más cercana a la parada, descartando el resto.

En cuanto a los formatos de las coordenadas, en el resto del proyecto se trabaja siempre con coordenadas UTM, pero los datos de las mediciones de lo ómnibus vienen en un formato similar a las coordenadas en latitud y longitud. Para poder trabajar con éstas coordenadas, se define una función de conversión que se detalla más adelante.

La función "analisisPorLinea"toma los tramos y los recorre en forma ordenada. Para cada tramo busca entre todos los registros de los viajes las mediciones más cercanas de cada viaje a cada parada. Se utilizan para ésto las funciones "imToUTM", la cual recibe como entrada las coordenadas en formato de latitud-longitud, y retorna las coordenadas en UTM, y la función "distancia"que retorna la distancia en metros entre cada par de coordenadas UTM. Si se tiene dos registros válidos de un viaje para un tramo, se miran los timestamp de ambas mediciones y se calcula la demora en segundos a partir de la función "tiempo". Luego se verifica que las mediciones se encuentren dentro del rango horario definido por el usuario, y de ser así se calcula la velocidad mediante la función "velocidad". Éste valor de velocidad se va acumulando en el tramo, y además se suma uno a la cantidad de mediciones del tramo, para calcular el promedio al finalizar el proceso.

Una vez que se procesaron todos los registros de todos los viajes se procede a hacer el cálculo del promedio de velocidad de cada tramo, el cual se hace dividiendo la velocidad acumulada entre la cantidad de mediciones para cada tramo. En los casos de tramos para los cuales no se cuenta con ningún registro válido, y por lo tanto no se tiene una velocidad definida, se le asigna la misma velocidad que al tramo anterior.

Finalmente, y también para cada tramo, se le asigna una demora promedio que se calcula en función de la velocidad. Dicha demora se presenta en segundos, dado que sus valores no son demasiado grandes. Una vez que se terminan los cálculos, se procede a guardar los datos escribiéndolos en un archivo CVS con el formato de salida ya definido. Se genera un archivo por cada variante de línea, coincidiendo el nombre del archivo con el código de la variante de línea a la que pertenece.

Conversión de coordenadas

La conversión entre coordenadas geográficas no es sencilla, por lo que se cuenta con un módulo que realiza los cálculos correspondientes.

Lo primero que se debe tener en consideración, es que el formato de las coordenadas de las mediciones aportadas por la IM no se corresponden en forma directa a ningún estándar. Se trata de coordenadas GMS (grados, minutos, segundos), pero escritas de forma poco usual, por lo que para poder trabajar con ellas se debe realizar una primera transformación al formato habitual. Esta primera transformación fue aportada por los tutores del proyecto, ya que cuentan con experiencia previa en el uso de estos datos.

En los datos de la IM se tiene una cadena de texto (numérico) con el formato "aabbccc", y para convertirlo al formato estándar de coordenadas GMS se debe aplicar la siguiente operación: $coordenadas = aa + (bb/60) + (ccc/60000)$. Dicha conversión debe realizarse tanto para las latitudes como para las longitudes.

Una vez que se tienen las coordenadas en el formato estándar de GMS, se pueden convertir al formato UTM para que coincida con el formato que se maneja en el resto de los datos provenientes del catálogo de datos abiertos.

Para la conversión de coordenadas GMS a UTM, se utiliza un algoritmo creado por Steven Dutch, docente de la universidad de Winsconsin, y adaptado por la universidad estatal de Montana, para disponibilizarlo en formato de código Javascript [30]. Dadas las similitudes entre el lenguaje Javascript y Scala, se realizó la conversión de las funciones para ser incorporadas en el proyecto.

El proceso de conversión de coordenadas cuenta con una serie de funciones auxiliares. La función principal es "LatLonToUTMXY"la cual recibe como parámetros de entrada las dos componentes de una coordenadas GMS, es decir, la latitud y la longitud. La salida es un arreglo de números reales, compuesto por las coordenadas UTM (x,y) calculadas y la zona. En este proyecto no se tiene en consideración la zona, ya que la ciudad de montevideo se encuentra compendida en una única zona UTM.

Funciones: "tiempo", "distancia", "velocidad"

Si bien los cálculos de diferencia entre dos marcas de tiempo, o la distancia entre dos puntos son cálculos sencillos, éstos son utilizados varias veces a lo largo del proyecto, por lo que es conveniente contar con funciones que resuelvan una única vez el problema para luego ser reutilizadas donde sea necesario.

Es así que se cuenta con una función "distancia", que dadas las coordenadas UTM de dos puntos retorna la distancia en metros entre ambos. Para esto hace uso del teorema de Pitágoras.

Por otro lado se cuenta con una función "velocidad", la cual recibe como parámetros el tiempo y la distancia, y lo que hace es realizar la división (ya

que velocidad = distancia / tiempo). En este caso las unidades de medida del tiempo y distancia son segundos y metros respectivamente, obteniendo la velocidad de metros por segundo. Para finalizar, la función realiza una conversión a kilómetros por hora, multiplicando el valor inicial por 3,6.

Finalmente se cuenta con la función "tiempo"que toma dos cadenas de texto, que representan los timestamp de los registros, y a partir de ellos calcula la diferencia en segundos entre ambos. Para ello se analiza la diferencia en horas, luego se calcula la diferencia en minutos y se le suma los minutos correspondientes en el caso que haya al menos una hora de diferencia. Para terminar se calcula la diferencia en segundos, sumando los segundos que se corresponden por la diferencia en minutos antes calculada.

6.1.3. Etapa 3

En la última etapa de procesamiento, se procede a unir los datos de todas las variantes de línea en un único archivo en formato CSV donde se contenga toda la información necesaria. Ésta etapa se realiza en forma secuencial, ya que se genera un único archivo de salida y no es conveniente tener varias escrituras concurrentes en el mismo archivo.

Lo que se hace en esta etapa es, para cada variante de línea, abrir el archivo que contiene sus tramos y copiar sus registros a un archivo de salida que es único para todas las variantes de línea. También se aprovecha esta recorrida para genera un resumen de los resultados obtenidos, escribiendo en otro archivo la cantidad de tramos sin datos detectados para cada variante de línea. También se incluye una velocidad promedio para la variante de línea que se está analizando y un promedio de la cantidad de medidas utilizadas para los cálculos en los diferentes tramos.

6.2. visualización de resultados

Una vez que culmina el procesamiento de los datos, se tiene un archivo CSV que contiene los datos que se deben mostrar a los usuarios. En primera instancia se deben transformar los datos geográficos expuestos como texto en geometrías con las cuales se pueda trabajar de mejor manera. También se debe buscar la mejor forma de almacenar esos datos generados y realizar la construcción de la herramienta de visualización de los datos.

6.2.1. Generación de datos geográficos y almacenamiento

Para la creación de las geometrías se utiliza la herramienta QGIS, con la cual se puede leer el archivo CSV y seleccionar los campos que se deben convertir. Esta herramienta genera las geometrías y permite también almacenarlas en archivos ShapeFiles o en bases de datos geográficas.

Para la generación de las geometrías, se tomó la decisión de identificar a cada tramo por un punto, el cual corresponde a la parada de destino del tramo en cuestión. El hecho de tomar puntos y no lineas o polilíneas responde a una limitación de QGIS, que solamente permite generar puntos a partir de archivos CSV.

El primer paso es importar el archivo con la opción "crear capa a partir de archivo de texto delimitado". Allí se abre una ventana sencilla en la cual se selecciona el archivo y las opciones de carga, en dónde se debe seleccionar cuáles son los campos que contienen las coordenadas que definen el punto o conjunto de puntos. Al hacer clic en "aceptar"se abre una segunda ventana para seleccionar las coordenadas en las cuales se encuentran definidos los puntos, que en el caso del proyecto es UTM 21s. Una vez que culmina la carga del archivo, se genera una capa en pantalla en la cual se pueden observar los puntos que se definieron anteriormente.

Para almacenar los datos, primero se debe contar con una base datos geográfica. Para el proyecto se realizaron pruebas con una base de datos local, para luego contratar un servidor con una base de datos PostGIS ya configurada. Una vez definida la base de datos, se utiliza la herramienta QGIS para exportar la capa de tramos a la base de datos, con la funcionalidad de exportación de datos. Es un proceso que lleva varios minutos dada la gran cantidad de datos que se manejan, pero que crea automáticamente la tabla en la base de datos y que almacena las geometrías siguiendo los estándares de la OGC.

Para la creación de la base de datos solamente es necesario instalar PostreSQL, el cual puede ser adquirido en forma gratuita a través de Internet. El mismo instalador se encarga al final de consultar si se desea instalar algún complemento, entre los cuales aparece el complemento de PostGIS. Allí se procede a la instalación, quedando una base de datos geográfica en cuestión de minutos.

6.2.2. Configuración de los servidores

La segunda capa de la arquitectura del proyecto está compuesta por dos servidores diferentes, los cuales tienen diferentes tareas y responsabilidades

dentro del proceso de exposición de los datos.

6.2.2.1. Servidor geográfico

El primero de los servidores es un servidor geográfico. En este caso se trata de un servidor Geoserver. El mismo se comunica con la base de datos de la capa inferior y expone los datos mediante webservices geográficos.

Al igual que sucede con la base de datos, en primera instancia se instaló Geoserver en forma local, para luego contratar un servidor que cuenta con Geoserver previamente instalado. Para el uso de Geoserver a nivel local alcanza con descargar el instalador desde internet, el cual se consigue en forma gratuita.

Una vez instalado Geoserver, se accede a la página de administración del servidor con los datos de usuario creados durante la instalación. En dicha página se deben proceder a realizar varias configuraciones. En primera instancia se debe configurar la nueva fuente de datos, que en el caso de este proyecto se trata de una base de datos PostGIS. Con la conexión a la base de datos creada se necesita comenzar a configurar cuales son las capas geográficas que serán expuestas a través del servidor.

Las capas configuradas en el servidor geográfico para este proyecto son dos: la capa de líneas, que contiene los recorridos de las diferentes variantes de línea, y la capa principal con los tramos y los valores de velocidades. En la primer capa se tiene una polilínea, donde cada línea del mapa representa un recorrido de una variante de línea de ómnibus. Por otro lado, la segunda capa es una mulipunto, donde cada punto representa el final de un tramo asociado a una variante de línea, con datos de velocidad y tiempo promedio, así como distancia total del tramo en cuestión.

Uso de SLD

El SLD presenta una oportunidad de mejorar la visualización de determinadas capas, permitiendo definir colores, o formato y tamaño del texto entre otras opciones. En este proyecto se aprovechan algunas de las posibilidades que brinda este estándar, modificando los criterios para la visualización de la capa de tramos, por tratarse de la más importante del trabajo.

En primera instancia se definen cinco rangos de velocidades, y se les asigna un color a cada uno de ellos, de forma de simplificar la visualización. Con esto se puede detectar fácilmente y con una simple mirada en el mapa, cuales son los puntos más conflictivos (con menores velocidades) en color rojo o aquellos en los que la velocidad es mayor en un color verde oscuro. En la Tabla 6.1 se especifican los rangos de velocidad y los colores utilizados.

Color	Rango de velocidades
Rojo	menos de 10km/h
Anaranjado	entre 10 y 15km/h
Amarillo	entre 15 y 20km/h
Verde claro	entre 20 y 25km/h
Verde oscuro	más de 25km/h

Tabla 6.1: Rangos de velocidades

Como se ha presentado previamente, la capa de tramos está formada por el conjunto de puntos de finalización de cada tramo, cuyo dato más relevante es la velocidad, por lo que se ha decidido utilizar SLD para que el dato de velocidad se vea junto al círculo que marca el fin del tramo.

Para el uso de SLD se define un archivo XML con el formato definido por el estándar. Lo primero que se define se trata de una capa con estilo de usuario, para luego definir las reglas. Se debe definir una regla por cada categoría o rango de velocidad definido, y en cada regla se determina un filtro, que considere los valores del atributo velocidad en este caso. En caso que se cumpla con los requisitos definidos, la figura seguirá los lineamientos que se definan para la regla.

En todas las reglas se definieron las etiquetas PointSymbolizer que indica la forma en que se dibujarán los puntos y TextSymbolizer que indica las propiedades del texto que acompañará al dibujo del punto. Para el primero se define el uso de un círculo, el color mediante código HTML hexadecimal y el tamaño mediante un número entero. Entre las propiedades del texto que se deben definir se encuentra la propiedad del punto que se desea mostrar, en este caso la velocidad, luego los datos de la fuente de texto y el color. En este proyecto se utiliza fuente Arial de tamaño 16 en negrita (bold) y de color negro. En esta etiqueta se puede definir la cantidad de dígitos decimales a mostrar, mediante el uso de una expresión regular. Para este proyecto se muestran dos dígitos decimales.

6.2.2.2. Servidor de aplicaciones

El servidor de aplicaciones es el de menor complejidad para su configuración, ya que se cuenta con una única página web y pocos archivos para alojar.

Luego de contar con un servidor web configurado y levantado, se necesita colocar el archivo HTML que contiene la página web principal en el directorio raíz del servidor para que la misma quede en funcionamiento. Es la propia

página web que realizará las llamadas al servidor geográfico mediante el uso de javascrip, como se detalla en la sección de cliente web.

Para finalizar la correcta configuración del servidor, es necesario crear los directorios para alojar los archivos del sitio web (con los datos fuente utilizados en el proyecto), y copiar dichos archivos a las ubicaciones que les corresponde.

6.2.3. Cliente web

La visualización de resultados se realiza mediante el uso de un cliente grueso, ya que la lógica y las llamadas al servidor geográfico se realizan mediante el uso de javascript del lado del cliente.

En esta capa se implementan por un lado la visualización de los resultados, y por otro se cuenta con una serie de filtros para seleccionar los datos que se desean mostrar. Finalmente se cuenta con una sección con información a través de la cual se puede acceder a las fuentes de datos utilizados en el proyecto, así como los archivos con los datos utilizados.

El cliente web está formado por un único sitio web con una única página. Dicha página web está creada a través de código HTML, con la utilización de Javascript. Se utiliza Bootstrap, para facilitar que la página web sea correctamente visualizada en cualquier dispositivo, siendo que ese framework garantiza que la página web se adaptará a las diferentes configuraciones de pantallas en las que se visualizará el sitio web.

Los datos se visualizan sobre en un mapa de la ciudad de Montevideo, obtenido de Open Street Map, el cual es gratuito y de uso libre. Sobre dicho mapa se agregan la capa de líneas donde se muestran los recorridos de las diferentes variantes de línea y la capa de tramos que fue explicada anteriormente.

El manejo del mapa y sus diferentes capas, así como el uso de los filtros se realiza a través de la librería de Javascrip conocida como OpenLayers. Dicha librería es la encargada de realizar las diferentes llamadas al servidor geográfico y de mostrar el mapa en pantalla.

En primera instancia, al cargar el sitio web, se realizan dos llamadas al servidor geográfico utilizando los webservices geográficos expuestos por dicho servidor. En particular se utiliza WMS, trayendo una imagen de cada una de las capas que se utilizan en el proyecto: capa de líneas y capa de tramos. Para la realización de dichas llamadas se alcanza con especificar el servidor y la capa que se desea levantar en cada llamada. Es opcional la utilización de filtros, los cuales se utilizan luego.

Una vez que se completaron las dos llamadas WMS al servidor geográfico, OpenLayers se encarga de generar el mapa con las tres capas, mediante el

uso de la función Map primero para la generación del mapa y de la función View luego para mostrar el mapa en el sitio web.

Existen varios parámetros que se pueden configurar en la función de visualización. En este proyecto, se optó por centrar la vista inicial sobre la ciudad de Montevideo, ya que los datos solo tienen sentido para dicha región. Es por eso se que realizan dos configuraciones: el nivel del zoom y las coordenadas en las cuales se centra el mapa. Para la elección de las coordenadas se utiliza el código EPSG, cambiando el valor por defecto (EPSG:4326) por las coordenadas de la ciudad de Montevideo (EPSG:3857). También se optó por un valor de en el zoom para que la visualización por defecto incluya la totalidad de los recorridos de lineas estudiadas.

6.2.3.1. Implementación de filtros

Para un mejor análisis de la información volcada sobre el mapa, se implementan una serie de filtros que permiten mostrar solamente los datos de determinada variante de línea, de determinado rango horario o de determinado mes.

Los filtros se organizan en la sección inicial, con una lista desplegable para cada filtro y tres checkboxes para elegir que filtros se deben aplicar. Luego se cuenta con un botón para aplicar los filtros y otro para borrar los filtros aplicados.

Las listas de de opciones para los filtros de rangos horarios y de mes se cargan estáticamente en el código html, ya que las opciones son limitadas. Por otro lado el filtro de variantes de línea tiene una lógica diferente. Se tienen dos listas desplegables para organizar de mejor manera la búsqueda de la variante de línea a mostrar. En la primera lista se selecciona el identificador de una línea de ómnibus y luego se selecciona la variante de línea en la segunda lista.

La lista de lineas de ómnibus se carga directamente desde un archivo de texto alojado en el servidor web, mientras que la segunda lista se carga en forma dinámica. A partir de la línea de ómnibus seleccionada, mediante el uso de Javascript se carga la segunda lista con los códigos de las variantes de línea que pertenecen a la línea seleccionada. Esto se realiza mediante la función *actualizarVariantes()* que actualiza los valores de la lista desplegable.

Una vez seleccionadas las opciones de filtro deseadas, al hacer clic en el botón de *aplicar filtros*, se realizan dos nuevas llamadas al servidor geográfico para traer las nuevas capas de variantes de línea y de tramos. La diferencia con las llamadas iniciales al WMS se da en el uso de un filtro CQL, el cual consta de una cadena de texto que se arma con Javascript a partir de las opciones seleccionadas. Luego OpenLayers actualiza las imágenes obtenidas para las capas que forman parte del mapa, que en este caso son la capa de

variantes de línea y la capa de tramos.

El formato del texto CQL es sencillo, ya que se forma con una cadena de texto. El formato es una concatenación de pares *campo-valor*, separados por el término *and*. En el lugar del campo se indica el campo de la capa que se desea filtrar y en el lugar del valor se coloca el valor seleccionado en la lista desplegable del filtro. Así por ejemplo una llamada con el filtro CQL para seleccionar los datos de la variante de línea 1 sería: *cod_varian = 1*.

Capítulo 7

Análisis experimental

En esta sección se exponen los resultados obtenidos del análisis experimental sobre el procesamiento de datos de movilidad de ómnibus de la ciudad de Montevideo durante el año 2015. Se cuenta con diferentes análisis preliminares realizados sobre los datos de entrada, planificados con el fin de obtener insumos para la toma de decisiones del proyecto. Se destaca principalmente el análisis de las diferentes configuraciones de paralelismo usadas en busca de la mejor opción para realizar el procesamiento masivo de los datos de entrada.

7.1. Preparación de datos de GPS

La primera parte del proyecto consiste en la preparación de los datos de entrada. Para ello se implementa un script en Python que recorre cada archivo de texto de la entrada y divide las líneas de texto en diferentes archivos, generando una serie de archivos CSV por cada viaje del mes con todos los registros de cada viaje. También se genera otra serie de archivos, teniendo uno por cada variante de línea, en el cual se registran los viajes correspondientes a la variable de línea en cuestión.

Se realizaron tres ejecuciones sobre el mismo conjunto de datos (se tomó el archivo de datos de enero), para definir el tiempo promedio en cada ambiente de ejecución disponible, para finalmente realizar una ejecución por cada mes de datos en al ambiente más adecuado a las necesidades del problema.

En la Tabla 7.1 se listan los tiempos de ejecución en los dos ambientes disponibles. El primero de los dos ambientes disponibles para ejecuciones constaba de un PC personal, con Windows 7 y procesador AMD de cuatro núcleos de 1.9GHz. Las ejecuciones que se realizaron no fueron aceptables dada la gran demora en culminar la ejecución. El segundo ambiente se trata

de Nébula, un computador potente perteneciente al cluster FING, la infraestructura de cómputo de alto desempeño de la Facultad de ingeniería. Nébula cuenta con 24 procesadores AMD Opteron(tm) 6172 de 2.1GHz. Cada procesador cuenta a su vez con 12 núcleos. En este caso, las ejecuciones demandaron menos de 4 horas, alcanzándose así tiempos aceptables para el pre procesamiento de los datos de entrada.

Ejecución	Tiempo en PC de escritorio (horas)	Tiempo en nébula (horas)
1	25:49	3:50
2	26:38	3:25
3	-	3:35
Promedio	26:13	3:37

Tabla 7.1: Ejecuciones algoritmo separador de viajes

A partir de los tiempos de ejecución obtenidos en la etapa anterior, se optó por continuar con las ejecuciones solamente en Nébula, obteniéndose tiempos de ejecución similares para los diferentes meses procesados, como se puede ver en la Tabla 7.2. Estos tiempos son considerados aceptables ya que el pre procesamiento de los datos de entrada se realiza una única vez para cada mes de datos.

Mes de ejecución	Tiempo (horas)
Enero	3:50
Febrero	3:30
Marzo	4:22
Abril	3:49
Mayo	4:00
Junio	4:03
Julio	4:15
agosto	3:44
Setiembre	4:10
octubre	4:13
Noviembre	3:56
Diciembre	3:49

Tabla 7.2: Ejecuciones por mes de algoritmo separador de viajes

7.2. Pruebas preliminares

Para la toma de algunas decisiones clave en el proyecto, se realizaron pruebas preliminares. Dichas pruebas fueron realizadas sobre los datos de una única línea de ómnibus y sus variantes.

Con estas pruebas preliminares se analizó el correcto comportamiento de los algoritmos desarrollados y utilizados, así como algunas primeras mediciones de tiempo en base a diferentes configuraciones de los ambientes de ejecución.

También se realizaron otras pruebas con el fin de determinar los horarios pico, a partir de los que se detectó que algunas variantes de línea se ven principalmente afectadas en algunas horas de la mañana mientras que otras lo hacen en horas de la tarde. Es así que se definieron dos horarios pico, uno matutino y otro vespertino.

7.2.1. Pruebas sobre datos de la línea 100

En principio se realizaron pruebas sobre una determinada línea de ómnibus para probar la correcta implementación de los algoritmos utilizados. La línea de ómnibus seleccionada fue la 100, que posee cuatro variantes de línea diferentes (227, 231, 237 y 242). Para cada una de esta variantes de línea se realizó un chequeo de los datos obtenido mediante los algoritmos utilizados en el proyecto, contra otras fuentes de datos externas.

Por un lado se verificaron los cálculos de las distancias, contrastando los datos obtenidos por el procesamiento, respecto a los datos que se obtienen de la medición que se obtiene al utilizar Google Maps. Es así que se tomaron varios tramos al azar, para verificar las distancias de dichos tramos, concluyendo que el proceso es fidedigno en tramos correspondientes a líneas rectas, y que en caso de tramos no rectos, la distancia sufre alteraciones respecto al recorrido real del ómnibus. Este punto implica una simplificación para la implementación del presente proyecto, quedando la posibilidad de mejorar la aproximación de las distancias como trabajo a futuro.

Para ejemplificar se pueden considerar los tramos 2 y 3 de la variante 227 de la línea 100. El tramo 2 (ver Figura7.1) es el que une las paradas 4042 (Buenos Aires esquina Bartolomé Mitre) y 4019 (18 de Julio esquina Convención). El resultado del algoritmo arroja 422 metros, mientras que la medición realizada con Google Maps indica una distancia de 450 metros. Esta diferencia es porque el recorrido no es recto, sino que se cuenta con varias curvas.

Por otra parte, para el tramo 3 (ver Figura7.2) que une las paradas 4019 y 4200 (18 de Julio esquina Paraguay), que es un tramo recto, el algoritmo

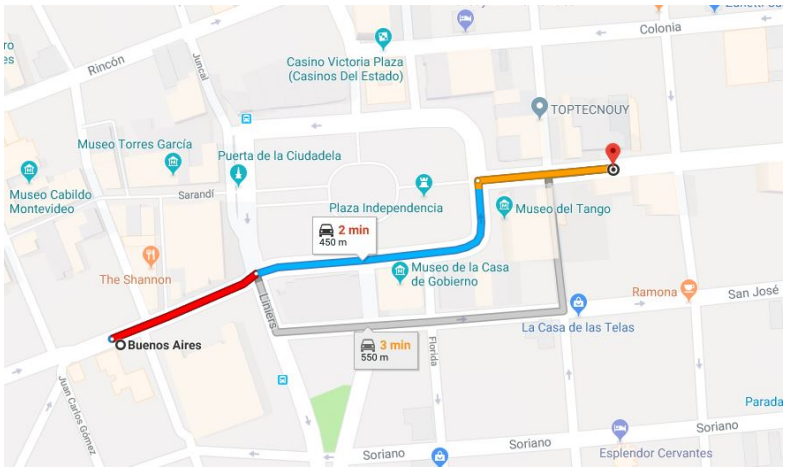


Figura 7.1: Distancia del tramo 2 según Google Maps.

indica una distancia de 401 metros mientras que Google Maps indica una distancia es 400 metros. Dado que la mayoría de los recorridos de las líneas urbanas son rectos, la estimación de los tramos como líneas rectas para el calculo de las distancias resulta aceptable.

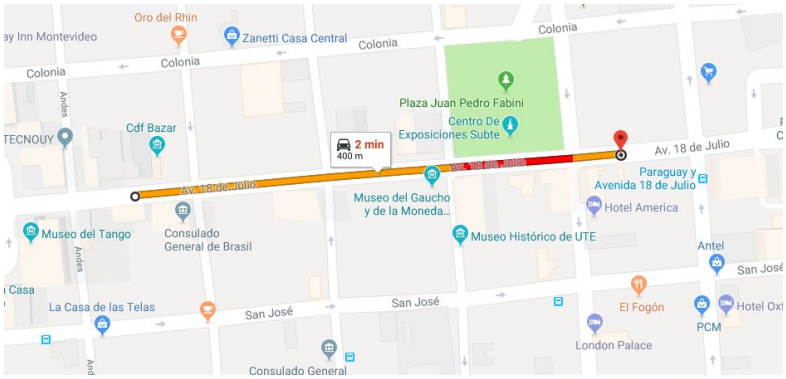


Figura 7.2: Distancia del tramo 3 según Google Maps.

7.2.1.1. Análisis de velocidades reales contra velocidades teóricas

Otra de las pruebas realizadas sobre datos de la línea 100, consistió en analizar que las velocidades calculadas sean razonables. Para este análisis se tomaron en cuenta los horarios por parada que se pueden obtener del catálogo de datos de la IM. Dicho conjunto de datos se corresponde a los horarios en los cuales cada ómnibus debe pasar por cada parada de su recorrido en forma teórica.

Por un lado se utilizan los datos obtenidos del procesamiento de datos de GPS, calculando la velocidad promedio de todas las variantes de línea correspondientes a la línea 100 (variantes 227, 231, 237 y 242). Por otro lado se implementó un script que analiza los horarios teóricos de las mismas variantes de línea, y calcula la velocidad promedio para la totalidad del recorrido.

Finalmente se realiza la comparación de las dos velocidades obtenidas, teniendo una velocidad teórica de 16,55km/h, contra una velocidad real de 18,73km/h. A raíz de la comparación realizada para la línea 100, se considera que el valor obtenido a partir de las mediciones de GPS no presenta un desvío significativo respecto a la velocidad promedio teórica.

La confiabilidad del algoritmo también se ve reflejada en la velocidad que se calcula para cada tramo, obteniéndose valores razonables. Las velocidades de la variante 227 de la línea 100 oscilan entre los 7km/h para el tramo más lento y los 32km/h en el más rápido para el promedio general de los viajes del mes de enero de 2015.

7.2.2. Determinación de horas pico

Para determinar las horas pico se realizaron ejecuciones, distinguiendo diferentes franjas horarias, para dos variantes de línea de la línea 100, que tienen recorridos opuestos. Es por eso que se tomó la variante 227 que va desde Plaza España hacia Bv Aparicio Saravia y la variante 231 que hace el recorrido opuesto. Cada franja horaria para la prueba corresponde a una hora reloj. Una vez realizadas todas las ejecuciones, se realizó el cálculo de la velocidad promedio del recorrido en cada franja horaria, detectando las tres horas de menor velocidad para cada variante de línea.

En la Tabla 7.3 se presentan los promedios de velocidad de la variante 227 de la línea 100 según los datos del mes de enero de 2015, de donde se desprende que en la franja horaria comprendida entre las 16hs y las 19hs es donde se presentan las velocidades más bajas. Esto es coincidente con los horarios que culminan la mayoría de las jornadas laborales, lo cual genera una carga mayor de tráfico de las líneas urbanas que se alejan del centro de la ciudad, como en este caso.

Por otra parte se puede observar la Tabla 7.4 en la cual se presentan las velocidades promedio de cada franja horaria para la variante 231 de la línea 100 según el mismo conjunto de datos de enero de 2015. En este caso se desprende que la franja horaria de menores velocidades se da entre las 6hs y las 9hs, coincidiendo con el comienzo de las jornadas laborales, momento en el que las líneas que van hacia el centro de la ciudad tienen mayor demanda por parte de los usuarios, como es el caso de esta variante de línea.

Cabe aclarar que en ambas tablas se excluyen las franjas horarias para

Horario	Promedio de velocidad (Km/h)
0 - 1	21,73
6 - 7	19,65
7 - 8	20,39
8 - 9	20,02
9 - 10	19,37
10 - 11	19,05
11 - 12	18,47
12 - 13	18,25
13 - 14	18,23
14 - 15	18,06
15 - 16	18,03
16 - 17	17,38
17 - 18	16,85
18 - 19	17,43
19 - 20	18,11
20 - 21	18,32
21 - 22	19,48
22 - 23	20,22
23 - 24	20,65

Tabla 7.3: Ejecuciones por hora - variante 227

las cuales no se cuenta con datos, que coinciden con horas de la madrugada en las cuales no hay servicios de las variantes seleccionadas.

A partir de las franjas detectadas se observa que la variante de línea 227 tiene menores promedios de velocidad en horas de la tarde, mientras que la variante de línea 231 presenta menores promedios de velocidad en los horarios de la mañana. Es por esto que se optó por trabajar con dos horarios pico diferentes, permitiendo al usuario verificar según la variante de línea el horario pico que sea más conveniente para el análisis.

Los horarios pico definidos constan de franjas horarias de tres horas de duración según lo observado en el análisis preliminar de los datos. El horario pico matutino se estima entre las 6hs y las 9hs, tomándose todas las mediciones comprendidas en dicha franja horaria, mientras que el horario pico vespertino es el comprendido entre las 16hs y las 19hs y para el cual se toman en consideración solamente las mediciones realizadas entre dichas horas.

Horario	Promedio de velocidad (Km/h)
4 - 5	19,24
5 - 6	18,76
6 - 7	17,77
7 - 8	16,62
8 - 9	16,94
9 - 10	18,01
10 - 11	18,08
11 - 12	18,19
12 - 13	18,32
13 - 14	18,58
14 - 15	18,54
15 - 16	18,48
16 - 17	18,60
17 - 18	18,53
18 - 19	18,60
19 - 20	19,18
20 - 21	19,61
21 - 22	20,19
22 - 23	21,83
23 - 24	20,79

Tabla 7.4: Ejecuciones por hora - variante 231

7.3. Ejecuciones con diferentes configuraciones de paralelismo

El foco principal del proyecto se basa en la capacidad de procesar grandes volúmenes de datos en tiempos de ejecución razonables, con el mayor aprovechamiento posible de las unidades de cómputo con las que se cuente.

En particular se utilizaron las instalaciones del cluster de la Facultad de ingeniería, infraestructura de cómputo de alto desempeño cuyo principal objetivo consiste en proveer soporte para la resolución de problemas complejos que demanden un gran poder de cómputo. Las tareas se realizaron accediendo a Nébula, computador de alto desempeño que forma parte del cluster. Se utilizaron diferentes configuraciones de paralelismo y se ejecutaron diferentes pruebas para intentar disminuir los tiempos de ejecución, analizando los valores de speedup obtenidos.

En primera instancia y como punto de partida se realizó una serie de ejecuciones utilizando una única unidad de cómputo, utilizando una versión serializada del algoritmo original. Luego se realizaron ejecuciones con la versión del algoritmo que incluye paralelismo, comenzando con una única unidad de cómputo, para comparar el comportamiento del algoritmo serializado con el comportamiento del algoritmo con paralelismo. Luego se fueron aumentando la cantidad de recursos de cómputo utilizados, tomando los tiempos de ejecución y realizando los cálculos de speedup para el promedio de tiempo de ejecución de cada configuración. Todas las ejecuciones se realizaron sobre el mismo conjunto de datos de entrada, utilizando los datos de enero de 2015.

nro. ejecución	algoritmo secuencial	algoritmo paralelo: unidades de cómputo						
		1	2	4	8	16	24	32
1	921	821	579	265	143	115	94	83
2	917	920	581	255	140	119	82	77
3	925	918	564	266	141	100	85	109
4	931	924	588	288	152	118	149	89
5	910	931	554	260	143	116	81	99
promedio	920,8	922,8	573,2	266,8	143,8	113,6	98,2	91,4

Tabla 7.5: Tiempos de ejecución (minutos) según cantidad de unidades de cómputo

Como se puede ver en la Tabla 7.5. y en la Figura 7.3. los tiempos de ejecución disminuyen rápidamente al aumentar la cantidad de unidades de cómputo. Sin embargo, también se puede observar que dicha disminución en el tiempo de ejecución es cada vez menor, presentándose una disminución de la eficiencia computacional al incrementar las unidades de cómputo. Esto se debe a las sinconizaciones internas que se realizan entre los procesadores, como se vio en la Sección 2.

En la Tabla 7.6. y en la Figura 7.4. se puede observar gráficamente el comportamiento del speedup. Se observa claramente que al aumentar la cantidad de unidades de cómputo el speedup va aumentando en forma sublineal.

El comportamiento de la eficiencia computacional se observa en la Tabla 7.6. y en la Figura 7.5. En este caso se puede observar la disminución gradual, con excepción del caso de cuatro unidades de cómputo, en la eficiencia computacional. De todas formas se ve que el caso de una única unidad de cómputo es el caso de mayor eficiencia computacional.

Analizando los diferentes gráficos y los valores de los tiempos de ejecución se puede concluir que un nuevo aumento de la cantidad de unidades

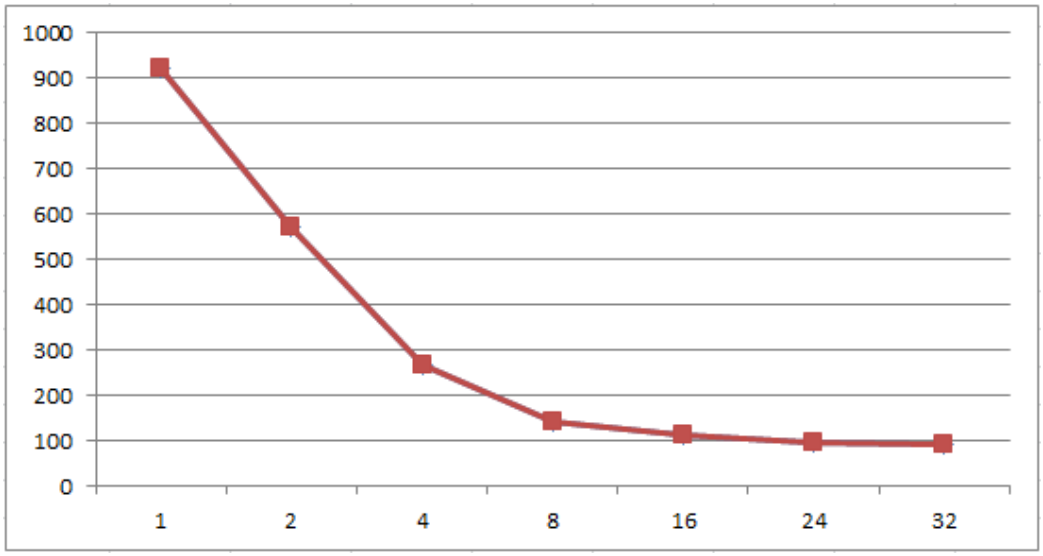


Figura 7.3: Tiempo de ejecución (en minutos) del procesamiento de datos.

de cómputo a usar no resultará en mejoras sustanciales en los tiempos de ejecución, dando lugar a su vez a peores valores de eficiencia computacional.

Unidades de cómputo	speedup	eficiencia computacional
1	1	1
2	1.61	0.80
4	3.46	0.86
8	6.42	0.80
16	8.12	0.51
24	9.40	0.39
32	10.10	0.32

Tabla 7.6: speedup y eficiencia computacional

Otro punto estudiado en el proyecto es el uso de la configuración por efecto que tienen las Scala Parallel collections, con el fin de determinar la utilidad de dicha configuración al problema del proyecto. Los valores de tiempos de ejecución son similares a los obtenidos con las ejecuciones que utilizan 24 y 32 unidades de cómputo. En particular se realizaron 5 ejecuciones sobre el mismo conjunto de datos utilizado en las pruebas anteriores (datos de enero de 2015), llegando a un tiempo de ejecución promedio de 95 minutos.

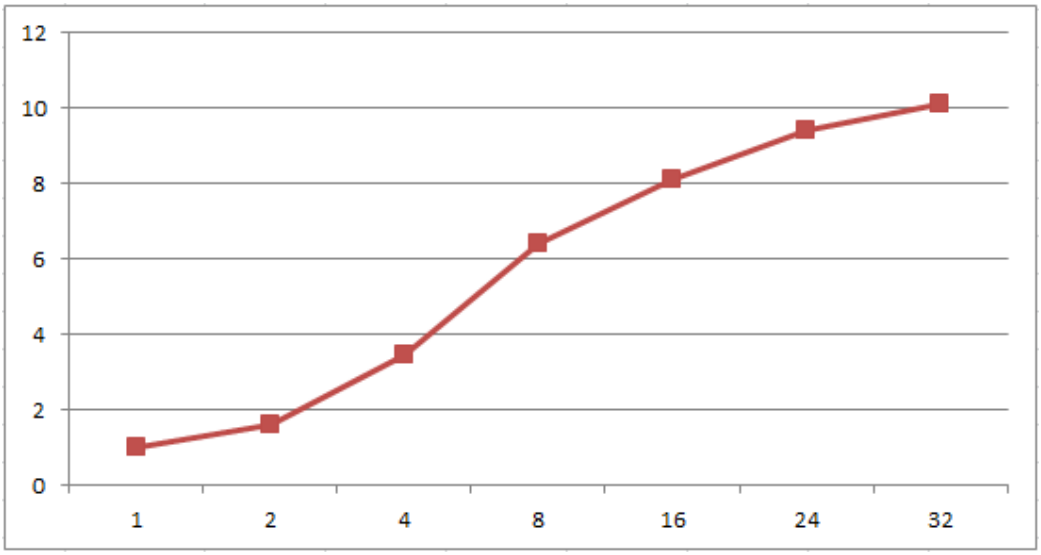


Figura 7.4: Valores de speedup.

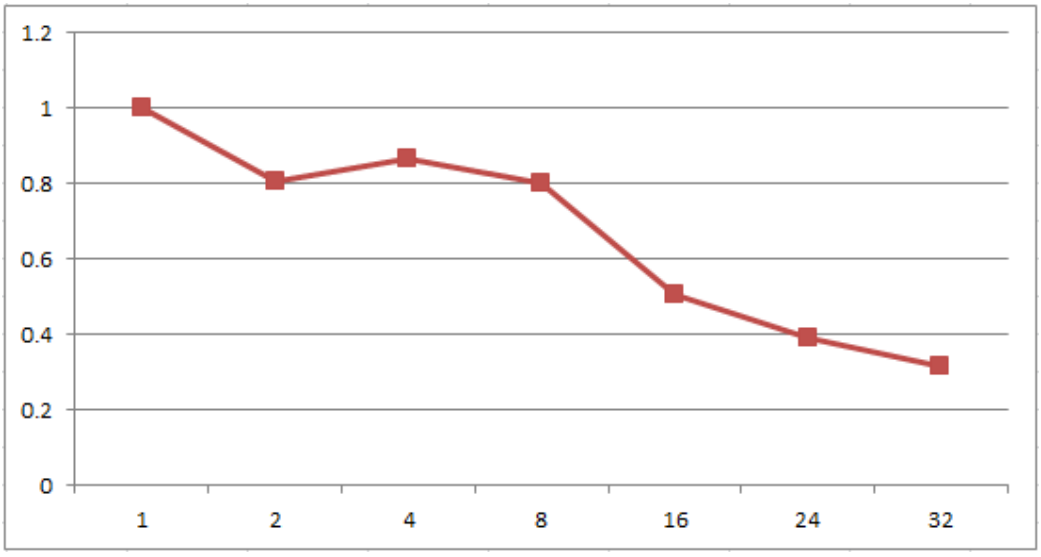


Figura 7.5: Valores de eficiencia computacional.

Capítulo 8

Conclusiones y trabajo a futuro

En esta sección se plantean las principales conclusiones del trabajo así como algunas opciones que pueden resultar de interés para un futuro desarrollo partiendo de los resultados del presente proyecto.

8.1. Conclusiones

Al culminar la realización del proyecto se pueden extraer varias conclusiones en cuanto a las diferentes áreas y tareas realizadas.

Por un lado, se llega a la conclusión inevitable de que el procesamiento de grandes volúmenes de datos debe ser realizado en forma paralela y que el poder de cómputo con el que se cuente puede resultar determinante para lograr tiempos de ejecución razonables y aceptables. En la etapa de pre-procesamiento se observó que con un computador de uso doméstico la separación de los datos de entrada insume un tiempo muy alto, el cual se logra mejorar al realizar la ejecución en un ambiente paralelo y con mayor potencia de cómputo.

Analizando la etapa de procesamiento, se observa que los tiempos de ejecución mejoran al aumentar la cantidad de unidades de cómputo disponibles, aunque también se debe considerar que la eficiencia computacional disminuye, por lo que es necesario encontrar un equilibrio entre las cantidad de unidades de cómputo que se utilizan y el aprovechamiento que se realiza sobre las mismas. Es así que se puede llegar a la conclusión de que una de las alternativas de configuración con tiempos de ejecución más bajos para el problema analizado y el hardware disponible se da al utilizar entre 24 y 32 unidades de cómputo. Sin embargo, el escenario de menor tiempo de ejecución puede variar en el caso de cambiar la infraestructura sobre la cual se ejecuta el algoritmo de resolución del problema, por lo que se deberá volver

a analizar los tiempos de ejecución y el speedup de ser necesario.

También se ha concluido que las Scala Parallel Collections resultan útiles para este tipo de problema, donde los datos tienen una partición bien definida, ya que simplifica enormemente la ejecución en paralelo del análisis de los datos. Otro aspecto que cabe resaltar es el buen comportamiento que tiene por defecto, logrando resultados óptimos sin necesidad de ningún tipo de configuración extra, resolviendo en forma automática la cantidad de unidades de cómputo a utilizar en base a las disponibles, realizando el balanceo de carga sin necesidad de manipular valores de configuración.

El uso de una base de datos PostgreSQL con complemento PotGIS fue satisfactorio, almacenando los datos en forma correcta e integrándose correctamente con el resto de las herramientas. Se destaca especialmente la integración con la herramienta QGIS, la cual fue fundamental para la conversión de datos numéricos en datos geográficos y su almacenamiento en la base de datos. Entre los puntos a destacar se encuentra la facilidad de uso, tanto para conectarse a la base de datos como para realizar la importación y exportación de datos.

Otra experiencia positiva fue el uso de Geoserver como servidor geográfico. Al igual que la base de datos utilizada, este servidor resultó fácil de instalar y utilizar. Se destacó la correcta integración con PostGIS y la facilidad que se presentó al momento de realizar la importación de las capas desde la base de datos. También resultó satisfactorio el uso de Geoserver, especialmente en la facilidad de uso de los webservices geográficos, los cuales se generan en forma automática al momento de realizar la importación de las capas desde la base de datos.

La herramienta utilizada para visualizar los resultados en un navegador web fue OpenLayers, la cual es ampliamente recomendada para este uso. La experiencia de uso resultó sumamente favorable, tanto en las posibilidades que brinda así como en el manejo de los web services geográficos, los cuales se integran perfectamente para poder visualizar los datos obtenidos a través de ellos de forma amigable al usuario. Se destacó la presentación obtenida utilizando esta librería de Javascript, logrando resultados vistosos y amigables hacia los usuarios finales.

En resumen se puede concluir que la elección de las herramientas utilizadas para el manejo de los datos geográficos, así como la presentación de los mismos a los usuarios finales fue correcta, pudiendo integrarlas con cierta facilidad a nivel de configuraciones, obteniendo buenos resultados a fin de exponer la información generada del proceso hacia los potenciales usuarios finales de la plataforma.

En cuanto al problema estudiado, se considera que el mismo fue atacado correctamente, logrando un análisis válido y de utilidad para la posible toma

de decisiones. Mediante el uso de programación paralela se logró resolver el problema de analizar un volumen grande de datos en tiempos que no superan las 10 horas, frente a un tiempo que superaba las 50 horas de ejecución con una configuración serializada, utilizando el mismo hardware para los datos de un mes puntual. Esto aplicado a la totalidad de los meses procesados, nos lleva a una estimación que indica que la reducción total de los tiempos de ejecución del análisis de los datos del presente proyecto disminuyó de 600 horas de cómputo a menos de 120.

Además de lograr el procesamiento de los datos en tiempos de ejecución razonables, el proyecto busca generar información relevante para su utilización en la resolución de problemas de optimización orientados a mejorar la movilidad y el transporte público. Sobre este punto se considera que el proyecto arroja resultados útiles, dando información resumida sobre el comportamiento del transporte público urbano de la ciudad de Montevideo.

Con la información generada en el proyecto, se facilita la toma de decisiones a los usuarios responsables, pudiéndose detectar rápidamente lugares problemáticos para el transporte urbano mediante la utilización de rangos de velocidad asociados a colores que se muestran sobre el mapa de la ciudad. También se permite profundizar el análisis, accediendo a detalles sobre los recorridos de las diversas líneas que forman parte del STM, lo cual puede resultar igualmente valioso.

Otra información relevante obtenida de este proyecto, es la determinación de las horas pico, en las cuales las líneas de ómnibus encuentran mayores dificultades de movilidad. El proyecto permite observar el comportamiento y detalle de los datos durante estos horarios.

Finalmente se puede concluir que, luego del procesamiento de los datos, se obtiene información valiosa y con valor agregado para realizar diferentes tipos de análisis, con el objetivo de realizar mejoras en el servicio prestado a los usuarios de transporte de la ciudad de Montevideo.

8.2. Trabajo a futuro

En este proyecto se han realizado una serie de simplificaciones, las cuales pueden se consideran válidas dado que las principales motivaciones estaban dirigidas a realizar mejoras en el procesamiento de grandes volúmenes de datos mediante la utilización de programación paralela y en la visualización de resultados mediante el aprovechamiento de las existencia de herramientas para el manejo de datos geográficos. Es así que se observan algunas posibilidades de mejoras, las cuales pueden ser abordadas en futuros trabajos que se encarguen de profundizar en el algoritmo utilizado para la conversión de

los datos de los viajes en datos de velocidades y tiempos promedio de las diferentes líneas de ómnibus del STM.

Actualmente se tiene una limitante en cuanto a la presentación de los datos de entrada, ya que los mismos fueron otorgados a la FING en un formato poco flexible. El hecho de que los datos vengan todos en archivos CSV unificados por cada mes de preprocesamiento lleva a la necesidad de incurrir en una etapa de pre procesamiento que se podría evitar si se contara con otras formas de presentación de los datos de entrada, como podrían ser bases de datos sobre las cuales realizar las consultas necesarias para la obtención de los datos en los formatos que sean más favorables para su posterior procesamiento. Es así que se ve como un posible desarrollo a futuro, el análisis de diferentes alternativas para la obtención de los datos de entrada, así como su implementación y la adaptación de este proyecto para quitar la etapa de pre-procesamiento. Eliminado este pre-procesamientos, se podría obtener una reducción cercana al 50 % en los tiempos de ejecución en total, ya que dicho porcentaje de tiempo es lo que demanda la etapa de pre-procesamiento.

Otro punto que fue simplificado en el proyecto es la medición de distancias de cada tramo en los recorridos de ómnibus. Este punto puede ser mejorado, especialmente en los tramos que no están formados por una única línea recta. Una alternativa para analizar en el futuro consiste en la utilización de los datos geográficos que se conocen del recorrido, los cuales se obtienen de los shapefiles disponibles en el catálogo de datos abiertos, a partir de los cuales se podría realizar una medición exacta de la distancia total de cada tramo. Al mejorar la estimación de las distancias, se mejoraría la estimación de los tiempos y velocidades promedio de cada variante de línea analizada.

El análisis de las posibilidades que brindan las Scala Parallel Collections también resulta interesante como una alternativa a profundizar en el futuro. Si bien la experiencia de su uso en el proyecto fue positiva, y se realizaron algunas pruebas sobre sus configuraciones por defecto, se considera que se puede realizar un análisis aún más profundo de sus capacidades y limitaciones.

Finalmente se considera que la herramienta generada para la visualización de los datos generados debe ser objeto de nuevas iteraciones incrementales, con el fin de aumentar la información que se brinda al usuario final para colaborar con la toma de decisiones. Es de especial importancia poner la herramienta a prueba con usuarios cercanos a la problemática, de preferencia candidatos a ser usuarios finales de la aplicación, con poder de toma de decisiones. De esta forma pueden aportar su conocimiento con el fin de generar un feedback que resulte en mejoras al aplicativo, para el mayor aprovechamiento del análisis los datos disponibles. Siempre poniendo el foco en que el fin del proyecto es mejorar la experiencia de los usuarios del transporte

metropolitano de la ciudad de Montevideo en este caso, o de cualquier ciudad del mundo en que se quiera aplicar.

Bibliografía

[1] Ing. Agrim. Edison Rosas. Conceptos introductorios. *Curso Taller de Sistemas de Información Geográficos Empresariales - FING*, 2017. Online: https://eva.fing.edu.uy/pluginfile.php/159225/mod_resource/content/1/GIS_ConceptosIntroductorios2017.pdf Fecha de acceso: 31/10/2017.

[2] Sergio Nesmachnow, Santiago Iturriaga, and Nestor Rocchetti. Introducción a la computación de alta performance. *Curso de HPC - FING*, 2017. Online: https://www.fing.edu.uy/inco/cursos/hpc/material/clases/Tema1-2017.pdf Fecha de acceso: 31/10/2017.

[3] Michael N Fienen and Randall J Hunt. High-throughput computing versus high-performance computing for groundwater applications. 2015. https://www.researchgate.net/publication/271530646_High-Throughput_Computing_Versus_High-Performance_Computing_for_Groundwater_Applications Fecha de acceso: 31/10/2017.

[4] Jorge Domínguez Chávez. ¿qué es computación de alto desempeño? (hpc). 2009. Online: https://www.researchgate.net/publication/236012148_que_es_computacion_de_alto_desempeno_HPC Fecha de acceso: 31/10/2017.

[5] Sergio Nesmachnow. Computación científica de alto desempeño en la facultad de ingeniería, universidad de la república. *Revista de la Asociación de Ingenieros del Uruguay 61 (1), 12-15*, 2010. Online: https://www.fing.edu.uy/biblioteca/revistas/396505.pdf Fecha de acceso: 31/10/2017.

[6] Ian Foster and Addison-Wesley. Designing and building parallel programs. 1995. Online: https://www.mcs.anl.gov/ itf/dbpp/ Fecha de acceso: 31/10/2017.

[7] Sergio Nesmachnow, Santiago Iturriaga, and Nestor Rocchetti. Programación paralela. *Curso de HPC - FING*, 2017. Online: https://www.fing.edu.uy/inco/cursos/hpc/material/clases/Tema3-2017.pdf Fecha de acceso: 31/10/2017.

[8] Sergio Nesmachnow, Santiago Iturriaga, and Nestor Rocchetti. Programación multithreading. *Curso de HPC - FING*, 2017. Online: https://www.fing.edu.uy/inco/cursos/hpc/material/clases/Tema6-2017.pdf Fecha de acceso: 31/10/2017.

[9] Martin Odersky, Philippe Altherr, Vincent Cremet, Gilles Dubochet, Burak Emir, Philipp Haller, Stéphane Micheloud, Nikolay Mihaylov, Adriaan Moors, Lukas Rytz, Michel Schinz, Erik Stenman, and Matthias Zenger. Scala language specification. Online: http://scala-lang.org/files/archive/spec/2.12/ Fecha de acceso: 31/10/2017.

[10] Aleksandar Prokopec and Heather Miller. Parallel collections. *Scala documentation*. Online: http://docs.scala-lang.org/overviews/parallel-collections/overview.html Fecha de acceso: 31/10/2017.

[11] Olivier Blanvillain and Heather Miller. Clases concretas de las colecciones paralelas. *Scala documentation*. Online: https://docs.scala-lang.org/es/overviews/parallel-collections/concrete-parallel-collections.html Fecha de acceso: 31/10/2017.

[12] Aleksandar Prokopec and Heather Miller. Configuring parallel collections. *Scala documentation*. Online: http://docs.scala-lang.org/overviews/parallel-collections/configuration.html Fecha de acceso: 31/10/2017.

[13] OGC. Simple feature access - part 2: Sql option. *OGC standars*, 2010. Online: http://www.opengeospatial.org/standards/sfs Fecha de acceso: 31/10/2017.

[14] PostGIS Project Steering Committee. Postgis 2.4.2dev manual. *PostGIS documentation*, 2017. Online: http://postgis.net/docs/manual-2.4/ Fecha de acceso: 31/10/2017.

[15] Geoserver user manual. 2017. Online: http://docs.geoserver.org/stable/en/user/ Fecha de acceso: 31/10/2017.

[16] Data management. *Geoserver user manual*, 2017. Online: http://docs.geoserver.org/stable/en/user/data/index.html Fecha de acceso: 31/10/2017.

[17] Services. *Geoserver user manual*, 2017. Online: http://docs.geoserver.org/stable/en/user/services/index.html Fecha de acceso: 31/10/2017.

[18] Web map service. *OGC standars*, 2006. Online: http://www.opengeospatial.org/standards/wms Fecha de acceso: 31/10/2017.

[19] Web feature service. *OGC standars*, 2014. Online: http://www.opengeospatial.org/standards/wfs Fecha de acceso: 31/10/2017.

[20] Web coverage service. *OGC standars*, 2010. Online: http://www.opengeospatial.org/standards/wcs Fecha de acceso: 31/10/2017.

[21] Styled layer descriptor. *OGC standars*, 2007. Online: http://www.opengeospatial.org/standards/sld Fecha de acceso: 31/10/2017.

[22] Documentation. *OpenLayers*, 2017. Online: http://openlayers.org/en/latest/doc/ Fecha de acceso: 31/10/2017.

[23] Basic concepts. *OpenLayers tutorials*, 2017. Online: http://openlayers.org/en/latest/doc/tutorials/concepts.html Fecha de acceso: 31/10/2017.

[24] Características. *Documentación de QGIS 2.14*, 2017. Online: http://docs.qgis.org/2.14/es/docs/user_manual/preamble/features.html Fecha de acceso: 31/10/2017.

[25] Stm - sistema de transporte metropolitano. *Sitio web STM*, 2017. Online: http://www.montevideo.gub.uy/transito-y-transporte/stm-sistema-de-transporte-metropolitano Fecha de acceso: 31/10/2017.

[26] Joana Maria Seguí Pons and Maria Rosa Martínez Reynés. los sistemas inteligentes de transporte y sus efectos en la movilidad urbana e interurbana. *Scripta Nova - revista electrónica de geografía y ciencias sociales*, 2004. Online: https://catalogodatos.gub.uy/organization/intendencia-montevideo Fecha de acceso: 31/10/2017.

[27] Richard J. Weiland and Lara Baughman Purser. Intelligent transportation systems. *Committee on Intelligent Transportation Systems*, 2000. Online: http://onlinepubs.trb.org/onlinepubs/millennium/00058.pdf Fecha de acceso: 31/10/2017.

[28] Florent Demoraes, Francis Bondoux, Marc Souris, and Hidalgo Núñez. Innovaciones tecnológicas aplicadas al transporte colectivo en quito. 2004. Online: http://www.redalyc.org/html/126/12633107/ Fecha de acceso: 31/10/2017.

[29] Intendencia de Montevideo. Catálogo de datos abiertos. 2016. Online: https://catalogodatos.gub.uy/organization/intendencia-montevideo Fecha de acceso: 31/10/2017.

[30] Steven Dutch. Convert geographic units. Online: http://www.rcn.montana.edu/resources/converter.aspx Fecha de acceso: 31/10/2017.

[31] Aleksandar Prokopec, Phil Bagwell, Tiark Rompf, and Martin Odersky. On a generic parallel collection framework. *Scala documentation*, 2011. Online: https://infoscience.epfl.ch/record/165523/files/techrep.pdf Fecha de acceso: 31/10/2017.